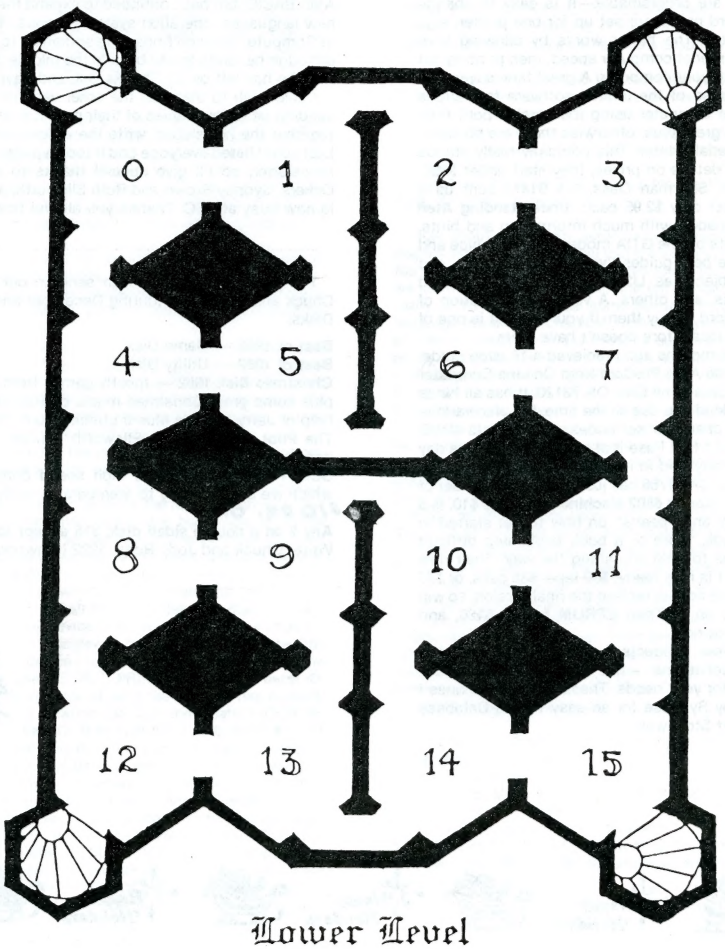
ATARI
COMPUTER
ENTHUSIASTS
3662 Vine Maple Dr. Eugene OR 97405

DECEMBER-JANUARY, 1983
Mike Dunn & Jim Bumpas, Editors

Castle Hexagon



Lower Level

News and Reviews

by M.R.Dunn, Co-Editor

Some very interesting products have been released for the Atari. O.S.S., (10379 Lansdale, Cuperinto, CA 95014) the originators of the Atari BASIC, BASIC A+, etc., have come out with a very ambitious new line of products for the Atari, and are now shipping them. They have two new types of DOS, OS/A+ ver.2 and ver.4 for double-density drives. The Version 4 allows you to use double-density, double sided, with various sector sizes so you can use it with an 8" or almost any drive. The version 2 can be used with either single or double density drives and is an improved version of the original OS/A+. Version 2 is included with all O.S.S. products, and version 4 is available from makers of Atari double density drives and controllers — see below.

Mac/65 is a new, very powerful and fast full Macro assembler with debugger, compatible with the Atari Assembler files. I expect this will become the standard of assemblers for the Atari at only $80.

TinyC at $100 and **C/65** (a C-compiler) for only $80 which runs with MAC/65 expand the Atari's language abilities to a new level. All are very well documented, and we could use full reviews of them by you out there who know enough about them to do so!

To go along with your new DOS, Software Publishers, Inc, 2500 E. Randol Mill Rd., Suite 125, Arlington, TX 76011 now has a double-density disk controller board, the **ATR8000,** allowing you to connect any 5½ or 8" drive to your Atari for $500, and for only $250 more, a 64K upgrade with CP/M!! The basic price includes a RS-232 and Centronics parallel ports, so you save the price of a 850 interface. The CP/M will read disks from Xerox, Bigboard, Cromemco, Kaypro, Osbourne and North Star. More in the next issue after we get one.

Also from Texas, Newell Industries, 3340 Nottingham In., Plano, TX 7507, who brought us the "Fast Chip" for the OS, now has a new OS board which allows you to put RAM or ROM and use the extra "hidden" 4K. $115 and up depending on the amount of memory, and a "SuperMon" monitor to put in one of the slots for $30.

Several months ago, I reviewed the **MicroCue** printer buffer by MicroCompatatible Corp. I was having some trouble with it, so sent it back to the company. They called me, and gave me the choice of getting it fixed, or waiting a while for their new model and getting a new one for free! I now have their new **SmartBuffer**. They have also changed their name and address: **Data-Match,** 3810 Oakcliff Ind. Ct., Doraville, GA. 30340. The buffers come in various models, and price depends on number and type of ports as well as amount of memory. Mine has serial and parallel ports, and you can input and output either. All the new models are programable — it is easy to change printer codes, so you can use a word processor set up for one printer, e.g., Atari 825, and use any other printer. The buffer works by allowing your computer to send the data to the printer at computer speed, then printing out while you are doing other things with your computer. A great time saver, and especially handy for the Atari because of the limited software for various printers. The ability to connect your computer using the parallel port, then connecting a serial printer is also a great plus; otherwise there are no word-processors allowing the use of a serial printer. This company really stands behind its products. Write them for details on prices, they start under $300.

Alfred Handy Guides, POB 5964, Sherman Oaks, CA 91413 sent us a collection of their guides which cost only $2.95 each. **Understanding Atari Graphics** is a very nice little book loaded with much information and hints, including various charts (in color), lots on the GTIA modes, etc. Very nice and very handy. There is also one of the best guides I've seen on VisiCalc, and others on Understanding such subjects as LISP, BASIC, PASCAL, Word Processors, DataBase Managements, and others. A very nice collection of booklets so inexpensive you can afford to buy them if your interest is one of the above. Call 800-292-6122 if your local store doesn't have them.

Speaking of handy guides, several months ago I recieved a 14 page guide, really a multiple folded card, called the Atari **ProCard** from On-Line Computer Centers OKC, 10944 A N. May Ave, Oklahoma City, OK 73120. It has all kinds of neat stuff — mostly tables of the kind you use all the time, but also various printer codes, error codes, Internal character set codes, etc. It costs $9.95, and I figured no one would buy it, but I find I use it at least 2 or 3 times a day when I'm using my computer and really find in invaluable!!

Elcomp, 53 Redrock Lane, Pomona, CA 91766 has just released a number of new products. **How to program your Atari in 6502 Machine Language** $10, is a nice 106 page book filled with hints and "pearls" on how to get started in 6502. Not a comprehensive text-book, more of a book explaining difficult concepts and giving hints and aides to help you along the way. Their new inexpensive wordprocessor, **ATEXT-1** is now ready, $30 tape, $35 cass. or $70 cart. I had a prelimery version but have not yet recieve the final version, so will wait until I do so before reporting on it. Their **EPROM burner** $180, and **EPROM Cartridge** for $30 are also out now.

For Christmas, I'll suggest a few products I use all the time and recommend to anyone without reservations — not to imply other similar products are not as good or better for your needs. These are just the ones I use the most. **Filemanager 800+** by Synapse for an easy to use Database system (see review this issue by Kirt Stockwell.

For word processors, I like **Text Wizard** for short articles, quick letters, etc, and **Letter Perfect ROM version** for multiple page essays and reports. The ROM version has the added benefit of being able to use any printer by making a user-defined printer handler, and is worth the extra money.

ValForth by ValPar is an excellent, complete FORTH with many options — I don't believe you can do any better.

The new **MAC/65** by OSS is a macro-assembler hard to beat at any price, but at only $80 including the excellent new OS/A+ version 2 DOS is an incredible bargain.

Paint by Reston is my favorite "Draw" program, even though you cannot save the pictures on a file and use them, as in the also excellent **MicroPainter** and **Graphics Master** by DataSoft.

The best BASIC is **BASIC A+** by OSS; I much prefer it to MicroSoft except when just copying a program from a MicroSoft listing.

Any of the **Tricky Tutorials** from Santa Cruz Educational Software.

Tach-master for measuring your disk drive speed and **DataLink** for your Modem are two programs by Tony Dobre and available from Swifty I use constantly. **The Atari Computer** by Lon Poole (Osborne/McGraw Hill, **Atari Games and Recreations** by Kohl et al (Reston), and **The BASIC Handbook 2nd Ed.** by David Lien (CompuSoft) are all excellent choices for the beginner to intermediate Atari owner, while the **Compute's! Second Book of Atari** is for the the more advanced user.

For hardware products, **Mosaic** memory boards seem to be among the best, with **Axlon** and **Tara** also very good. For printers, I have had or used the Centronics 737, 739, Epson, MPC and others, but my present printer, the **IDS MicroPrism** is by far the best in print quality and features. The **Percom Disk drives** are very worthwhile — our bulletin board has been using 2 of them 24 hours a day for several months without one hitch. The **SmartModem** by Hayes is the nicest modem I have used.

There are also many other very good products, but the above are the ones I use all the time, so are on the top of my mind. All the companies mentioned are well known to me, either by meeting the heads at the various computer faires, visiting their companies or talking to them on the phone and all are very helpful and contantly trying to improve their products. It is interesting to note many of the products above were also recommended last year, although many are now in improved form.

The most helpful person of all to the Atari community has got to be **Bill Wilkinson** of OSS, whose company not only started it all with the original Atari BASIC, but has continued to expand the capabilities of the Atari through new languages, operation systems, books, and through his fantastic articles in Compute! He even finds time somehow to answer questions on the phone, although he tends to call back in the middle of the night because it is the only time he has left over!! Thanks Bill, and have a nice Holiday Season.

I also wish to thank all the other vendors who have been so generous in sending us review copies of their products and to all the people who help put together the newsletter, write the articles and produce such fine programs. Last year I listed everyone and it took a page; this year it would take the whole Newsletter, so I'll give special thanks to our most prolific authors: Stan Ockers, Sydney Brown and Ruth Ellsworth, and of course, Marc Benioff, who is now busy at USC. Thanks you all and have a nice holiday season!!

.....

For your pleasure, you can send to our program exchange chairpeople Chuck and Jody Ross (during December and January only) for the following Disks:

Best of 1982 — Game Disk
Best of 1982 — Utility Disk
Christmas Disk 1982 — mostly games from the Newsletter from July to Dec plus some great Christmas music by Ron and Aaron Ness, written with the help of Jerry White's Music utilities from PDI.
The Pilot Disk by Ruth Ellsworth —filled with educational programs and demos written in Pilot
SCOPY 810, our licensed high speed copy utility from Alliance Software which we can sell only to members. A really great and useful program.

$10 ea. OR
Any 2 on a double sided disk, $15 except for SCOPY 810
Write: Chuck and Jody Ross, 2222 Ironwood, Eugene, OR 97401.

BURST IO

by Pat Warnshuis, Editor, Portland Atari Club Newsletter,
3116 S.W. Evelyn St., Portland, OR 97219
(Reprinted from the June, 1982 issue)

(The P.A.C. is one of my favorite Newsletters, especially the monthly "Toolkit" section with its outstanding utilities. Pat is not only an excellent editor, he is also a talented programmer and a wonderful person. The following should be be very useful for you. —MD)

BURST IO (input-output) is an extremely fast method of transferring block data from cassette or disk to the computer memory or from the computer memory to your storage device. Burst IO (aka block IO) is not supported by ATARI or Microsoft BASIC. BASIC supports either single string transfers or the GET and PUT for single bytes of data. However, to save a graphics 8 screen to disk using PUT in a FOR . . . NEXT loop requires some three minutes. Using Burst IO you can dump a screen to disk in less than three seconds.

Where will you want to use Burst IO? Anywhere you have a large block of data to move in or out of memory. For example, you can save and retrieve any screen in any graphics mode. You can save, load or edit complete character sets. You can predefine player-missile data, then save and load them using Burst IO. You can load contiguous blocks of data or complete data base management files. If you saw the introductory program for the PAC program library disks with the PORTLAND ATARI CLUB banners flashing across the screen, then you saw an application of Burst IO. In that routine the one thousand bytes of player-missile data was loaded from the disk so fast nobody even noticed the disk was turned on!

First, a little background. (I'm going to talk about writing to a disk. The procedures are the same for writing to a cassette.) When BASIC writes to the disk, it only writes in segments of 128 bytes. It never writes a single byte as is implied by the PUT command. When you issue a PUT command, BASIC takes that single byte and adds it to an IO buffer of 128 bytes. The buffer is merely a part of your memory which is dedicated as a temporary accumulator where BASIC collects your individual bytes for an eventual sector write operation. (A sector is a block of 128 bytes on the disk or a record of 128 bytes on the cassette.)

BASIC keeps adding individual bytes to the buffer until the buffer is filled. When the buffer is filled, it writes the contents of the entire buffer to the disk in one operation. The only way you can write a partially filled buffer to the disk is to CLOSE a file or END a program. This is why you must CLOSE a file or END a program to get the last segment of your data saved.

Similarly, when you do a 'GET #3,A' BASIC does not read a single byte from the disk. Rather, on the first GET command, it reads an entire sector of 128 bytes into the IO buffer. Then it sets a pointer to the first byte in the buffer and is ready to move single bytes from the IO buffer to where you want them. When the pointer goes beyond 128 bytes, BASIC knows it must read in the next complete sector and reset its pointer to the start of the IO buffer before it can do the next GET instruction.

In BURST IO, however, BASIC and the operating system do not use the IO buffer. Instead, you simply tell the Central Input-Output (CIO) routine in the operating system where the data starts in memory and how many bytes to write. BANG! The operating system moves the data out in 125-byte chunks directly from memory to the disk. (I'll explain why it's 125 byte chunks instead of 128 byte chunks later.) Reading is even simpler in burst IO. You tell the CIO to read 65,000 bytes of data and tell it where to start storing the bytes in memory. When it gets to the end of the file, it stops reading.

Let's say it again, for emphasis: You simply tell the CIO where to start saving or loading the data and how many bytes to transfer.

The operating system does the transfer and returns the status of the operation, either A-OK or an error number, in a memory location which can be PEEK'd by your program.

How do you do it? Well, first OPEN a file for read or write. BASIC then sets up a block of control data for the file. You can have eight files open at a time. BASIC keeps track of what you are calling each file, where the buffer for each file is (or where the start location is for Burst IO), whether it is a read or write operation, the number of bytes involved, and some hardware-peculiar information which you tell it about using the AUX1, AUX2 bytes in your OPEN statement.

Oh, yes! It also keeps track of where the machine language routine is for getting in and out of the particular hardware which you are accessing: cassette, disk, printer, modem, CRT screen, etc. That's why you can talk to all of your hardware devices the same way. CIO only keeps track of WHERE the hardware driver is, not how it talks to the specific device. When the boys in the back room at ATARI wrote the rest of the operating system, they wrote the machine language routines for getting in and out of the standard devices, such as the disk, cassette, printer, etc. Then they made a table, called a Device Handler Table, which tells the CIO where each handler is. You can add your own handler routines and add them to the table. For example, if you have an odd-ball printer, or teletype machine, or a digitizer pad, or another computer, or any device at all, you patch in your code for talking to the device and add an entry in the device handler table to tell the CIO where the code is. Then you talk to the device the same way you talk to a disk, the CRT, or a printer. That is, you simply use GET/PUT, PRINT #/INPUT #, or burst IO.

BASIC needs 16 bytes of data to keep track of each file which it has opened. For our purposes, FILE 0 data starts at location 834. Call the location IOCB (Input/Output Control Block) for File 0. Then IOCB #1 is at IOCB + 16, IOCB #2 is at IOCB + 32, etc. At our IOCB# + 0 we POKE the direction command for our transfer: 7 for read, 11 for write, 12 for append. At IOCB# + 2,3 we POKE the start address for the data. IOCB# + 6,7 gets how many bytes we want to transfer. That's all!

Then we do a little machine-language call routine to jump directly into the operating system's CIO routine. This is the string you see so often which looks like this: BURSTIO$ = "hhh*LVd". The '*' and 'd' are in inverse video. Once the IOCB# data is set up, we initiate a burst IO by the statement: "XFER = USR(ADR(BURSTIO$),16*CHAN#)", where CHAN# is the file number we used in our OPEN statement for the file. The BURTSIO$ routine moves the CHAN# to the hardware index register and jumps to the operating system's CIO routine which does the burst IO.

Let's take another look at the screen editor which we ran in the June newsletter as an example of saving or loading text screens.

```
1 DIM F$(15):OPEN #3,4,0,"K:":OPEN #2,12,0,"E:":POKE 709,8:POKE 710,194
2 ? "Read or Write ":GET #3,C:IF C=82 THEN A=4:W=0:GOTO 4
3 ? "NEW or Add ":GET #3,C:A=8+(C=65):W=1
4 ? "FILE ";:INPUT F$:OPEN #1,A,0,F$:IF A=4 THEN 8
5 ? "} ± C Creates next screen":? " ± E Exits now":GET #3,C:? "} ± S Saves
screen";:IF C=5 THEN 12
6 GET #3,C:IF C
19 THEN PUT #2,C:GOTO 6
8 C=PEEK(560)+256*PEEK(561)+4
9 POKE 850,A+3-(A=9):POKE 852,PEEK(C):POKE 853,PEEK(C+1):POKE
856,192:POKE 857,3
10 C=USR(ADR("hhh*LVd"),16):IF W THEN 5 (* and d in reverse video!)
11 GET #3,C:IF PEEK(851)=1 THEN 8
12 ? "#":END
```

LN 1 opens the keyboard and screen editor so we can create a text screen. It also gives us the ever popular green screen.
LN 2,3 find out if we want to read or write a screen. An "R" returns an 82 so we set A = 4 for READ in our open statement. We also set a flag W = 0 which we use later to indicate we are not writing successive screens. If the keyboard returns anything other than an 'R', then we set A = 8 for a write and set the write flag to W = 1. If the keyboard input is 'A' for APPEND, we set A = 9 for the OPEN statement. (Remember, this code is cryptic because I'm trying to squeeze a complete screen editor into less that 1K bytes.)

LN 4 OPENs FILE #1 for our burst IO. So the start of our IOCB for this file will be 834 + 16, or location 850.
LN 5 issues the prompt to remind the user of the control characters used to create a screen, edit a screen or save the screen.
LN 6 demonstrates the power of the Atari operating system. This single line allows us to create an entire text screen using all of the editing features of the OS screen editor: enter, backspace, delete, insert, move the cursor, etc.
LN 8 finds out where the pointer for the start of our screen data is so we can use the location in our busrt IO.
LN 9 sets the IOCB command for a read or write, depending on our input in **LN 2,3.** POKE 852,PEEK(C):POKE 853,PEEK(C + 1) set the start location of our screen data. POKE 856,192:POKE 857,3 tell the IOCB how many bytes to transfer (The BASIC manual tells us a Graphics 0 screen has 960 bytes. (3x256 + 192.).
LN 10 initiates the burst IO to either read or write a screen to disk. If we are writing screens, we go back to create the next screen.
LN 11 waits for a keyboard input if we are reading screens. Following any keyboard input, we go read the next screen if the CIO returned a 1 for a status code. This means A-OK. If it runs into an end-of-file, we get a different code and fall through to **LN 12.**
LN 12 clears the screen, and closes the file. If your disk has MENU on it, use RUN "D:MENU" instead.

How about saving any screen in any graphics mode, including modified display lists? OK. First, recall that the screen display data is saved right at the top of memory. Location 106 always points to the top of memory. (Sometimes we lie to the operating system and tell it that top of memory is lower than it really is. This lets us steal some memory from BASIC for our own machine language routines or character sets or player-missile data, etc.) The very top of memory holds the text window for our screen if we used one. Then the rest of the screen data is saved below the window. Finally, the display list is saved immediately below the screen data. So-o-o-o-o-o-o. Why not save the display list along with the screen data? This way we can handle modified display lists also. (Thank you, Sheldon Leemon, from the Michigan ACE!)

If we look at the pointer in 106, we have the start of our data for burst IO. If we subtract the start of the display list from the top of memory, we have how many bytes to transfer. Might as well save all of the color registers and the graphics mode also. We can do this by discrete PUTs before we start the burst IO.

When we retrieve our screen, the complete graphics mode will be set up the same as when we saved the screen. Better save the contents of 106 also. (If we're writing a program for another machine, we should tell our program the top of memory is as low as we can and still fit our program in. That way, our program might run on machines with less memory than ours.)

Here's one approach to a general purpose screen save/load routine:
Note we are going to save the display list also. Since the display list points to where the screen data is, we must first POKE 106 with the lowest memory which still permits our program to run. This will make the saved display list compatible with smaller machines. If we get this general purpose, we will do well to save and restore the contents of 106 also. **LN 32500** saves the screen; **LN 32510** retrieves the screen.

```
1 ? "}D:SCRSAVE.LST":? :? "Save any mode screen to disk.":? "Load any
mode screen from disk.":?
2 ? "(Saves all colors + display list.)":? :?
3 ? "GOSUB 32500 to save screen.":? "GOSUB 32510 to load screen.":? :?
"Send FILE$ = D:filename.DAT"
4 ? "MODE = Graphics mode of screen"
5 ? "Must POKE 106,128 to be compatible with 32k machines.":? :STOP
10 REM D:SCRNSAVE.LST Save/retrieve any mode screen with display list &
all color registers.
```

```
32500 RAMTOP=PEEK(106)*256:DLIST=PEEK(560)+256*PEEK(561):
BYTES=RAMTOP-DLIST:HI=INT(BYTES/256):LO=BYTES-HI*256
32502 CLOSE #5:OPEN #5,8,0,FILE$:PUT #5,MODE:PUT #5,PEEK(560):PUT
#5,PEEK(561)
32503 FOR II=704 TO 712:PUT #5,PEEK(II):NEXT II:IOCB=834+5*16
32504 POKE IOCB,11:POKE IOCB+2,PEEK(560):POKE
IOCB+3,PEEK(561):POKE IOCB+6,LO:POKE IOCB+7,HI
32506 SCRSAVE=USR(ADR("hhh*LVd"),5*16):CLOSE #5:RETURN (reverse *
and d)
32510 CLOSE #5:OPEN #5,4,0,FILE$:GET #5,MODE:GRAPHICS MODE:GET
#5,A:POKE 560,A:GET #5,A:POKE 561,A
32511 FOR II=704 TO 712:GET #5,COLR:POKE II,COLR:NEXT II
32512 IOCB=834+5*16:POKE IOCB,7:POKE IOCB+2,PEEK(560):POKE
IOCB+3,PEEK(561):POKE IOCB+6,255:POKE IOCB+7,255
32514 SCRLOAD=USR(ADR("hhh*LVd"),5*16):CLOSE #5:RETURN (reverse *
and                                                    d)
```

The idea when retrieving a screen is to replicate the graphics status of the machine when you saved the screen.

Now you can generate, edit, save and retrieve PM data or character sets. Go modify those FONDEDIT or SUPERFONT programs to eliminate the DATA statements and save the complete character sets directly. Go ahead and save strings longer than 256 bytes. You know the address of the string from the ADR function and how many bytes it takes from the LEN funcion so you're all set for BURSTIO! Machine language routines for page six will load in an instant. Once you get them entered the first time, simply burst them out to a disk file for future use. Speedy. Elegant!

```
10 REM ************************
20 REM *  A DEMONSTRATION OF *
30 REM *PSYCHOLOGICAL TESTING*
40 REM *                      *
50 REM *   LINT HUTCHINSON    *
60 REM *                      *
70 REM ************************
80 REM
90 REM
100 REM ******************
110 REM *MAJOR VARIABLES*
120 REM ******************
130 REM
140 REM * NAME$      LAST NAME OF INDIVIDUAL
150 REM * D(300)     ARRAY OF 300 TEST TEST ITEMS
160 REM * SEX$       SEX (M OR F)
170 REM * RECORD$    INDIVIDUAL'S RECORD THAT RESULTS (
R,T,Z SCORES) ARE STORED IN
180 REM * ITEM$      TEST ITEMS THAT THE INDIVIDUAL RES
PONDS TO
190 REM * DATA$      NAME OF FACTOR SCALE CREATED
200 REM * CHECKED    NUMBER OF TEST ITEMS CHECKED
210 REM * A          DATA STATEMENT INDICATING SPECIFIC
 TEST ITEM USED IN CREATING A FACTOR SCALE
220 REM * L          LENGTH OF RECORD$
230 REM * RSM,SD     USED IN CORRECTING FOR NUMBER OF T
EST ITEMS CHECKED
240 REM * JJ,KK      TOTAL NUMBER OF INDICATIVE AND CON
TRAINDICATIVE DATA TEST ITEMS USED TO CREATE
FACTOR SCALES
250 REM
260 REM *****PROGRAM*****
270 DIM NAME$(30),D(300),SEX$(1),RECORD$(14),ITEM$(10)
,DATA$(20)
280 OPEN #1,4,0,"D:TABLE"
290 OPEN #2,4,0,"K:"
300 ? CHR$(125):POSITION 10,7:? "LAST NAME":POSITION 1
0,9:? "SEX"
310 POSITION 20,7:INPUT NAME$:POSITION 20,9:INPUT SEX$
:? CHR$(125)
320 RECORD$(1,2)="D:":RECORD$(3,10)=NAME$
330 L=LEN(RECORD$)
340 IF L<>10 THEN RECORD$((L+1),(L+1))="X":GOTO 330
350 RECORD$(11,14)=".SCO"
360 OPEN #3,8,0,RECORD$
370 ? #3;NAME$;CHR$(155);SEX$
380 CHECKED=0: POKE 752,1
390 FOR I=1 TO 300
400 ? CHR$(125):INPUT #1;ITEM$
410 POSITION 10,12:? " I AM ";ITEM$
420 POSITION 10,13:GET #2,U
430 IF CHR$(U)="Y" THEN D(I)=1:CHECKED=CHECKED+1:GOTO
450
440 D(I)=0
450 NEXT I
460 ? #3;"NUMBER CHECKED =";CHECKED:POKE 65,0:POKE 55
9,0
470 IF CHECKED>20 AND CHECKED<=48 AND SEX$="M" THEN OP
EN #4,4,0,"D:CONV.??"
480 IF CHECKED>73 AND CHECKED<=110 AND SEX$="M" THEN O
PEN #4,4,0,"D:CONV.??"
490 IF CHECKED>110 AND CHECKED<=138 AND SEX$="M" THEN
OPEN #4,4,0,"D:CONV.??"
500 IF CHECKED>138 AND CHECKED<=225 AND SEX$="M" THEN
OPEN #4,4,0,"D:CONV.??"
510 JJ=25:KK=13:GOSUB 550
520 JJ=19:KK=0: GOSUB 550
530 CLOSE #1:CLOSE #2:CLOSE #3:CLOSE #4
540 POKE 559,34:POKE 752,0:END
550 FOR J=1 TO JJ
560 READ A
570 R=R+D(A)
580 NEXT J
590 IF KK=0 THEN 640
600 FOR K=1 TO KK
610 READ A
620 R=R-D(A)
630 NEXT K
640 INPUT #4;RSM,SD
650 Z=INT((R-RSM)/SD)
660 T=50+10*Z
670 READ DATA$
680 ? #3;DATA$,"T-SCORE =";T,"Z-SCORE =";Z,"RAW =";
R
690 R=0:T=0:Z=0
700 RETURN
710 DATA 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,
19,20,21,22,23,24,25
720 DATA 42,47,69,81,124,141,142,143,200,201,227,236,2
44
730 DATA FIRST SCALE .....
740 DATA 2,6,7,11,11617,32,43,61,74,79,92,111,123,125,
200,201,210,220
750 DATA SECOND SCALE.....
```

NOTE: If this were an actual program, much more care would have to be taken to mask keys and trap errors. No attempt has been made to do this in this demonstration. —Lint

Here is a demonstration of how you can create a program to display and evaluate psychological tests. In fact, any test or set of test items which requires a YES or NO response can use this or a similar format. This program will not run as is. It is designed only as an example of techniques which can be used. This demonstration will:

1. Show how to use a one-dimensional array to store responses
2. Show how factor scales (dimensions) can be built using data statements of specific test items
3. How to interface conversion scales to correct raw scores

LINES 280-310

Open #1 is the file [D1:TABLE] that contains the actual test items [item$] which will be displayed on the screen for individual evaluation. Each test item can be evaluated as either applying "Y" or not applying "N" to the individual. The type of test items displayed on the screen could be: shy, happy, sad, impulsive, etc...

Open #2 is to the keyboard so the individual responding to the test item will not have to hit RETURN.

After this has been taken care of, the individual's last name [name$] and sex [sex$] are determined and inputed.

LINES 330-380

The file which will store the individual's records, #3, is created using his/her last name [name$]. If the last name is less than 8 characters (10 including D:), X's are added so an extension ".SCO" can be attached. This procedure is not necessary but allows you to selectively print only files with the ".SCO" extension.

The name and sex are then put in this file as identification. The CHR$(155) on **LINE 370** allows the variables to be put in the file in separate fields by generating an EOL character.

LINES 390-450

The loop **I** is from 1 to 300 - 300 being the total number of test items contained in file D1:TABLE. If you only have 10 test items to be evaluated, the loop will be from 1 to 10. These test items, to which the individual responds, are displayed one at a time on the screen. The test item is correctly positioned on the screen with the additional words:

I AM (and then the test item)

Line 420 waits for a response. If the response is "Y" then two important events take place at **LINE 430**:

1. The ARRAY D at location I D(I) is given a value of 1 if a "Y" is inputed
2. The number of test items checked is increased by 1

Notice: The ARRAY D at **Line 270** is DIM D(300). It is no mere coincidence the number of test items in D:TABLE also equals 300. and the loop **I** also is equal to 300. A Jungian might consider it synchronicity but you and I know better. The ARRAY D(300) is being used as a place holder. Every time an individual indicates a test item with a "Y" response, the number "1" is dropped into the array.

EXAMPLE: If the 5th test item [item$] in the file D:TABLE is "strange" the screen will display on the 5th time:

I AM STRANGE

The [I] variable in the loop for I = 1 to 300 (**LINE 390**) also is equal to 5, it being the 5th time through the loop. If the response is "Y" to this test item (CHR$(U) = "Y"), then the value of the ARRAY D at D(I) will equal 1 or D(5) = 1. This way we have indicated the 5th test item was responded to with a "Y" and the value of D(5) is 1.

If the response is anything else than "Y" then D(I) will equal 0 or in this case D(5) = 0 at **LINE 440**. We know for certain the individual response was not a "Y" to this test item. If you decide to leave **LINE 440** out of your program the test items not responded to with a "Y" will be filled with garbage. Garbage in . . . garbage #$•&•'@ out . . . Thank you, Fritz. You might even consider the possibility of filling the ARRAY D(300) with 0's before any response is asked of the individual!

In this demonstration, the number of test items checked [checked] is important in determining which conversion file to use (#4 D:CONV.??), **LINE 430** increases the value of [checked].

LINE 460 turns the screen off for faster calculations, turns the sound off and enters the number of test items checked in the individual's file.

LINES 470-500

These lines use two variables [checked] and [sex$] to determine which conversion file to open (#4,4,0,"D:CONV.??)". In reality the extension ?? could be anything 1m,1f,2m,2f,3m,3f, . . . indicating the file 1,2,3,4 and if the contents are male (m) or female (f).

LINES 510-520

Variables [JJ] and [KK] together indicate the total number of test items used to construct a factor scale (such as dominance, submissiveness etc.) In this demonstration, the first factor scale is dependent on a total of 38 test items - JJ + KK. The data statements [A] indicate which specific test items make up this first factor scale, i.e., test items 1,2,3,4 . . . 25 **LINE 710** and test items 42,47,69 . . . 244 in **LINE 720** — a total of 38 test items. The raw score [R] for this 38 test item first factor scale is the total of each test item's value (1 or 0) as determined by the individuals input ("Y" = 1, "N" = 0")

[JJ] is the total number of indicative test items and KK is the total number of contraindicative test items making up the first factor scale. The [JJ] test items (taken from the first 25 data statements) are added together and then the [KK] value is subtracted from this, resulting in the raw score [R] for the first factor scale. There are 25 data statements with values added [JJ = 25] and 13 data statements with values subtracted [KK = 13].

LINES 550-670

The loop [J] (J = 1 TO JJ) indicates how many data statements will be read at **LINE 560**. At **570** the raw score [R] is increased in the following manner:

R = R + D(A)

Up to this point [R] = 0 and [A] is the first data statement A = 1 at **LINE 710**.

```
R = 0 + D(A)
R = 0 + D(1)          1- specific test item #1
R = 0 + 1             1- value of specific test item #1
R = 1
```

[R] is equal to the value found in the ARRAY D at location 1 which in reality, is test item #1, or the first test item displayed on the screen for the individual to evaluate. If the value of D(1) is equal to 1, then we know the individual checked test item #1 with a "Y". It also increases the value of [R] raw score for the first factor scale by 1.

If the value stored in D(1) = 0, this indicate, the individual had checked test item #1 with "N". The value of [R] in this case does not increase:

```
R = R + R(A)
R = 0 + D(1)          1- specific test item #1
R = 0 + 0             0- value of specific test item #1
R = 0
```

Once the total number of indicative data statement test items [JJ = 25] has been processed, **LINE 590** checks to see if there are any contraindicative data statements test items to be calculated for the first factor scale. If there are contraindicative data statement test items [KK].

LINES 600-630 will read the data statement test items and subtract them from the raw score [R] already calculated for this factor scale. When this is completed, [R] is passed on for corrections and conversion.

LINES 640-700

Since the proper conversion scale has already been determined and opened, 2 variables appropriate to converting the raw score of the particular group are pulled out. Raw score mean [RSM] and standard deviation [SD] are pulled from the disc. The raw score [R] is then converted so the influence of the number of test items checked [checked] is no longer an influence in the final score.

The name of the factor scale is read [data$] from the data statements (in this case FIRST SCALE.) and this along with the [T], [Z], and [R] values are stored in the individual's file.

LINE 690 clears the values of [R], [T], and [Z] and the program returns from this subroutine to **LINE 510**.

LINE 520 indicates the total number of indicative data test item statements [JJ = 19] to be read, and the number of contraindicative data statements test items [KK = 0] (none in this case) for the next factor scale.

The program repeats the procedure of calculating the raw score [R], [T], and [Z] scores for the second factor scale and returns to **LINE 540** when completed. If there were more factor scales to be calculated, their [JJ] and [KK] values would be placed next.

Since this demonstration only has 2 factor scales (FIRST SCALE and SECOND SCALE), the program then closes all open files, turns the screen back on, and ends the program.

BUILD YOUR ATARI A TABLE

by Steven Berg, Philipine Islands

My Atari 800 had been sitting on my coffee table for far too long. It was time for a new home with a reasonable price. One can build this table for the Atari and accesories with one sheet of 3/4"plywood and little else.

Begin by cutting the plywood into the sizes shown in figure 2. Glue the 20" length of F1 to the 20" length of F2. Allow to dry for 24 hours. I will describe the construction as if you are facing the table as shown in figure 1.

If you want the rear of piece B to be flush with sides A then reduce its width from 23½ to 22¾ inches. If you have an MX-80 size printer, then cut the hole for the paper beginning 2" from the front and 4" from the right side of piece B. Cut the hole 1" in width and extend it to the left 12" so as to make a 1x12 inch slot. Move 3" to the left of the left edge of the paper slot and cut a second hole 2" from the front 1" in width and extend it to the left 3". The second hole is for your electrical connections.

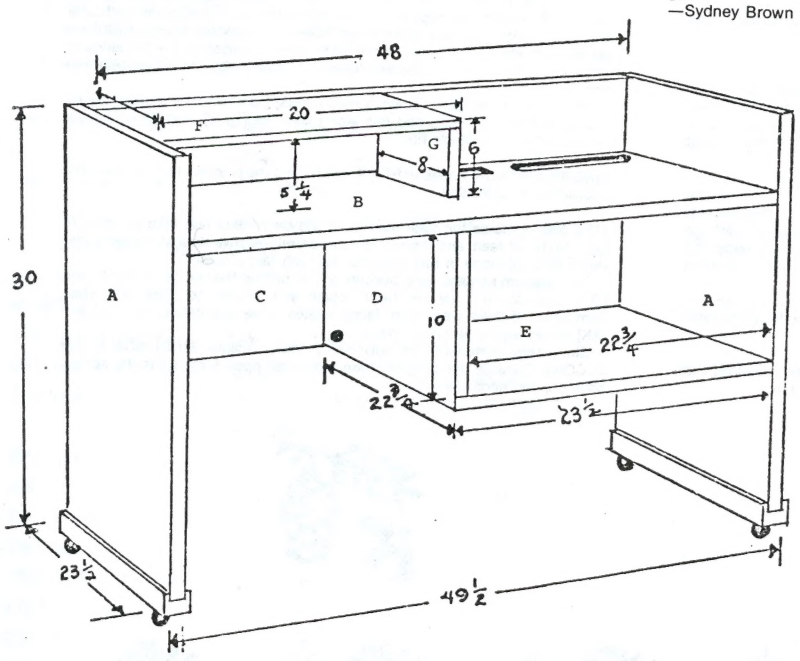
If you have an MX-100 size printer, shorten side A (right) by 6" so that side A (right) will be flush with piece B and begin the paper slot described above 2" from the right edge of piece B instead of 4" and extend it 14".

Nail and glue piece C to B and then nail sides A to B & C. Cut a hole for the power cord on the lower front of piece D. Nail and glue side E to piece D. Be sure not to do the reverse (nail D to E) or you will not have enough room to stack two disc drives on the shelf formed by the two pieces.

Nail and glue D & E to C, B and A (right). Cut 1/4" from the height of piece G (to make it 5¼"). Nail and glue piece F (F1 & F2) to piece G making sure there will be a 2" gap between G and C when it is attached on the table. Nail and glue piece F,G to pieces A & C (F only) and B (G only).

Cut a ¾ by ¾ inch groove in two 23½ inch long 2 x 2's. Place the bottoms of the side A's in the grooves and nail the 2 x 2's to the bottoms of the sides. Drill the appropriate hole for your casters in the 2 x 2's and install the casters.

I used a ¼ x ¾ veneer for the plywood edges. An alternative is wood veneer tape. Recess all the furniture nail heads with a punch and fill all the holes with wood putty. Sand the table. Stain as desired and rub it with tung oil. I used 1½" furniture nails, brads for the veneer and six 2" furniture nails to attach the 2 x 2's. You may want to use one of your extra pieces of plywood to separate the computer paper on the shelf from the electrical power supplies behind the disc drive which sits on the left side of the shelf. I placed the power supplies for my 800 and my 850 interface below the shelf for my monitor and between piece C and my 800.



CUTTING PATTERN ON OUTSIDE COVER

CASTLE HEXAGON

Castle Hexagon uses Mode 4 graphics. Notice in particular the detail of the bricks. If you type the program in be sure to notice the characters in strings in lines 124-170 are really control characters. You should press CTRL and the appropriate letter when typing these lines.

If you try to figure out how the program works, it might help to know the passage numbers between rooms can be found by summing the room numbers. The passages blocked are found in line 212. Passage #23 between rooms ÷20 and #3 is blocked and is opened when a match is made in room ÷3 (1st char. represent 0). The additional passages in line 216 are always blocked.

The sound routine is in VBI and as such is separate from the Basic program. If you make a match and the tune starts playing, you can continue on while the music plays (interrupted if you go to another room). The routine is slightly different from the one in "Vultures." It uses pure notes and has a pause between notes. The pause length is controlled by a POKE to 1779.

The ending routine cycles through various Mode 4 characters. If you want to see it without playing the game, insert: 305 GOTO 600.

—Stan Ockers

SCRAMBLED EGGS

Instructions: You and your shipmates are marooned on an island which is nothing but rock. The only food available is seagull eggs.

The seagulls are too scared to land as you might take to them so they lay their eggs while still flying around in the clouds.

You must try to catch the eggs without breaking them. Breaking 12 will end the game.

Program Description: Lines 1-6 define all strings for the shape of the man, the eggs, and the broken eggs for his head.

Lines 10-55 set up the pointer for the string used for PM Graphics.

Lines 56-101 put up the display and game name used in the display mode.

Lines 102-195 initialize and set up the player missile graphics, colour and width registers. They also put the players and missiles in their original positions.

Lines 197-199 set up the varialbes to their starting values and the program is ready to start.

Lines 200-299 contain the main program loop. In this loop resides most of the main functions such as direction control and movement of the eggs.

Lines 300-304 randomly choose the missile in which the next egg appears.

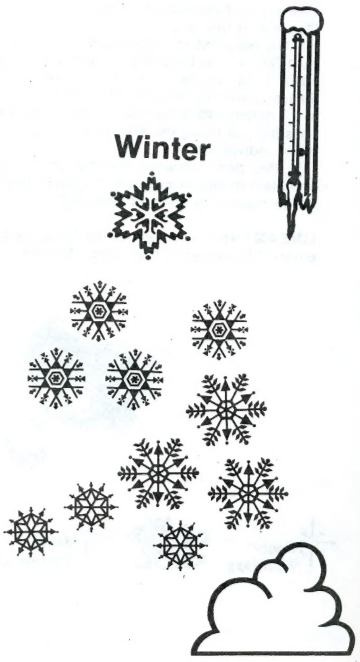
Lines 310-343 are used to draw the broken eggs in **GR.7** on the ground. Also the broken egg sound is produced here.

Lines 350-360 print the score and check to see if "game over" has been reached.

Lines 361-380 contain the game over routine and the timing for the automated display routine.

Lines 990-999 are used to clear the PM area and draw the scenery for the game.

—Sydney Brown



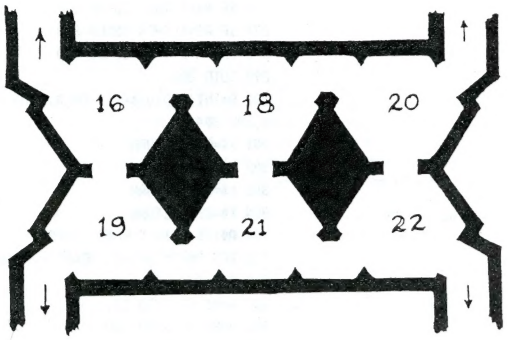
E G G S by Sydney Brown

Another fine game from Australia-but look out for Lines 6,310,340,368,and 700-they have graphic charactors

Use the REM chr$ #'s to enter the unprintable charactors for the machine language subroutes

```
0 REM ACE NEWSLETTER 3662 VINE MAPLE EUGENE OR 97405:LOOK FOR GRAPHIC CHARACTORS ON LINES 6,310,340,368,700
1 ? "}":HI=0:REM by Sidney Brown
5 DIM P$(1),M$(37),B$(5),Y$(3),W$(7),E1$(2),E2$(2),E3$(2),E4$(2),Z$(73),X$(2)
6 M$=" ( ($       44((((((((($$$$$$$$f "
:B$="ffff$":Y$=" ( ":W$="B   z2 ":E1$="  ":E2$="  ":E3$="00":E4$="  "
7 REM M$=chr$$  24,(,24,(,$,189,25,153,255,255,127,44((((,0,((($$$$$$$$f,231:Y$=chr$$ 24,(,12
8 REM W$=chr$$ B,247,255,254,z2 :E1$=chr$$ 3,3:E2$=chr$$ 12,12:E4$=chr$$ 192,192
10 VX=PEEK(134)+256*PEEK(135):UX=PEEK(140)+256*PEEK(141)
50 GRAPHICS 2:POKE 712,8:POKE 708,254:POKE 53248,0:POKE 53249,0:POKE 53250,0:POKE 53251,0:POKE 53252,0:POKE 53253,0
55 POKE 53254,0:POKE 53255,0:? #6;" S C R A M B L E D     E G G S"
60 FOR W=1 TO 40:? #6;"@@@@";:NEXT W
66 ? :? " PRESS START BUTTON TO START THE GAME";
90 POKE 53279,0
95 IF PEEK(53279)<>6 AND PEEK(77)<14 THEN 95
100 GRAPHICS 7:POKE 82,0:POKE 559,46:POKE 752,1:? "       SCRAMBLED  EGGS  !!":?
101 ? " USE STICK 1 AND MOVE MAN TO CATCH EGGS       MISSING 12 EGGS ENDS THE GAME";
102 POKE 704,40:POKE 705,254:POKE 706,14:POKE 707,202:POKE 712,2
104 POKE 708,14:POKE 709,254:POKE 710,34:POKE 711,14:POKE 623,17
106 PX=PEEK(106)-24:POKE 54279,PX:POKE 53277,3
108 POKE 53256,3:POKE 53257,0:POKE 53258,0:POKE 53259,1:POKE 53260,85:POKE 53255,1
110 POKE 53249,1:POKE 53250,1:POKE 53252,1:POKE 53253,1:POKE 53254,1
120 OX=256*PX+384-UX:BX=INT(OX/256):AX=OX-256*BX
121 POKE VX+2,AX:POKE VX+3,BX:POKE VX+4,113:POKE VX+5,2:POKE VX+6,113:POKE VX+7,2
190 GOSUB 990
195 P$(330,332)=Y$:P$(459,465)=W$:P$(204,208)=B$:P$(588,624)=M$
197 H1=100:H2=160:H3=110:H4=120:POKE 53252,H1:POKE 53253,H2:POKE 53254,H3:POKE 53255,H4:I1=1:I2=-1:I3=1:I4=-1
198 M=142:C=0:N=20:EC=0:EM=0:GOSUB 350
199 POKE 53251,M:POKE 53248,134
200 POKE 53278,0
201 IF PEEK(53279)=6 THEN 100
210 ST=STICK(0):IF (ST=9 OR ST=10 OR ST=11) AND M>50 THEN M=M-4
211 IF (ST=5 OR ST=6 OR ST=7) AND M<190 THEN M=M+4
212 POKE 53248,M-8:POKE 53251,M
240 IF EC/5=INT(EC/5) AND AC=1 THEN N=N-1:AC=0:IF N<5 THEN N=5
241 IF EC/5<>INT(EC/5) THEN AC=1
250 C=C+1:IF C>N THEN C=0:GOSUB 300:P$(19,20)=X$
251 Z$=P$(17,88):P$(19,90)=Z$
252 H1=H1+I1:POKE 53252,H1:IF H1<55 THEN I1=1
253 IF H1>195 THEN I1=-1:FOR W=22 TO 0 STEP -2:SOUND 0,W,10,10:NEXT W
254 H2=H2+I2:POKE 53253,H2:IF H2<65 THEN I2=1
255 ST=STICK(0):IF (ST=9 OR ST=10 OR ST=11) AND M>50 THEN M=M-4
256 IF (ST=5 OR ST=6 OR ST=7) AND M<190 THEN M=M+4
257 POKE 53248,M-8:POKE 53251,M
260 IF H2>155 THEN I2=-1:FOR W=22 TO 0 STEP -2:SOUND 0,W,10,10:NEXT W
261 H3=H3+I3:POKE 53254,H3:IF H3<85 THEN I3=1:FOR W=22 TO 0 STEP -2:SOUND 0,W,10,10:NEXT W
262 IF H3>125 THEN I3=-1:FOR W=22 TO 0 STEP -2:SOUND 0,W,10,10:NEXT W
263 H4=H4+I4:POKE 53255,H4:IF H4<75 THEN I4=1:FOR W=22 TO 0 STEP -2:SOUND 0,W,10,10:NEXT W
264 IF H4>165 THEN I4=-1:FOR W=22 TO 0 STEP -2:SOUND 0,W,10,10:NEXT W
265 P5=PEEK(53248):P6=PEEK(53249):P7=PEEK(53250):P8=PEEK(53251)
266 IF P5>3 THEN GOSUB 330
267 IF P6>3 THEN GOSUB 332
268 IF P7>3 THEN GOSUB 334
269 IF P8>3 THEN GOSUB 336
270 P1=PEEK(53256):P2=PEEK(53257):P3=PEEK(53258):P4=PEEK(53259)
271 IF P1=8 OR P1=9 THEN GOSUB 390
272 IF P2=8 OR P2=9 THEN GOSUB 390
273 IF P3=8 OR P3=9 THEN GOSUB 390
274 IF P4=8 OR P4=9 THEN GOSUB 390
275 IF P1=1 THEN GOSUB 380
276 IF P2=1 THEN GOSUB 380
277 IF P3=1 THEN GOSUB 380
278 IF P4=1 THEN GOSUB 380
299 GOTO 200
300 R=INT(RND(0)*4+1):ON R GOTO 301,302,303,304
301 X$=E4$:RETURN
302 X$=E3$:RETURN
303 X$=E1$:RETURN
304 X$=E2$:RETURN
310 P$(75,84)="":POKE 53278,0
311 REM P$(75,84)=10 HEARTS(chr$$ 0's)
319 RETURN
330 A=H1-52:GOTO 339
332 A=H2-52:GOTO 339
334 A=H3-52:GOTO 339
336 A=H4-52:GOTO 339
339 B=RND(0)*10+66
340 SOUND 0,6,8,14:P$(80,90)="":COLOR 1:PLOT A+2,B+1:DRAWTO A+8,B+1:PLOT A+1,B+2:DRAWTO A+9,B+2
341 REM P$(80,90)=10 hearts
342 PLOT A,B+3:DRAWTO A+6,B+3:PLOT A+2,B+4:DRAWTO A+4,B+4:COLOR 2:PLOT A+4,B:DRAWTO A+6,B:PLOT A+3,B+1
343 DRAWTO A+6,B+1:PLOT A+3,B+2:DRAWTO A+5,B+2:SOUND 0,0,0,0:EM=EM+1:POKE 53278,0:GOTO 350
350 SOUND 0,0,0,0:? "}":? :? :? "EGGS SAVED--EGGS LOST---------HI SCORE  ";:? EC,EM,:? ,:? HI;
355 IF EM>11 THEN 360
359 POKE 77,0:RETURN
360 IF EC>HI THEN HI=EC:GOTO 350
361 ? "      G A M E   O V E R ":? "PRESS START BUTTON TO RESTART GAME";
365 POKE 53248,134:POKE 53251,142:POKE 53279,0
366 IF PEEK(77)>7 THEN 50
367 IF PEEK(53279)<>6 THEN 366
368 COLOR 3:FOR W=66 TO 79:PLOT 0,W:DRAWTO 159,W:NEXT W:FOR W=20 TO 90 STEP 10:P$(W,W+9)="(10 hearts)":NEXT W
369 GOTO 195
380 EC=EC+1:SOUND 0,21,10,10:GOSUB 350:GOTO 310
390 SOUND 0,2,8,14:FOR W=1 TO 20:NEXT W:POKE 53249,M+4:POKE 53250,M+4
391 FOR W=180 TO 246 STEP 3:SOUND 0,W,12,4:NEXT W
399 SOUND 0,0,0,0:FOR W=1 TO 300:NEXT W:POKE 53249,1:POKE 53250,1:EM=EM+1:GOSUB 350:GOTO 310
700 E1$="    ":F2$="    ":E3$="    ":F3$="    ":E4$="@@@@":F4$="    "
701 REM E1$=chr$ #4,4,4,4:F2$=chr$$ 0,12,12,0:E3$=chr$ #16,16,16,16:F3$=chr$ #0  0:F4$=chr$ #0,128,128,0
990 FOR W=PX*256+384 TO PX*256+1023:POKE W,0:NEXT W
991 COLOR 3:PLOT 0,65:POSITION 0,79:POKE 765,3:XIO 18,#6,0,0,"S:"
992 COLOR 1:PLOT 14,0:DRAWTO 100,0:POSITION 0,7:POKE 765,1:XIO 18,#6,0,0,"S:":PLOT 21,8:DRAWTO 77,8
993 PLOT 100,8:DRAWTO 121,8:PLOT 35,9:DRAWTO 49,9:PLOT 135,0:DRAWTO 159,0
999 RETURN
```

Castle Hexagon



Upper Level

```
1 REM ********************************
2 REM **      ACE NEWSLETTER        **
3 REM **   3662 VINE MAPLE DR       **
4 REM **    EUGENE, OR 97405        **
5 REM **     DEC/JAN ISSUE 82/83    **
6 REM **           $10 YR           **
7 REM ********************************
10 REM *********************
20 REM ** CASTLE HEXAGON **
30 REM **  S.O.  10/82   **
40 REM *********************
45 GRAPHICS 18:POSITION 6,4:? #6;"Ca5t
LE":POSITION 6,6:? #6;"HexAgOn"
49 REM * Sound generation in VBI, Data
 in page 6 *
50 DIM VBI$(75):RESTORE 52:FOR J=1 TO
75:READ A:VBI$(J,J)=CHR$(A):NEXT J
52 DATA 174,252,6,240,67,173,244,6,240
,32,206,244,6,208,57
54 DATA 189,0,6,141,0,210,10,234,234
56 DATA 141,2,210,232,189,0,6,240,30,1
41,248,6,232,142,252,6,208,30,206,248,
6,16,25,169,0
58 DATA 141,0,210,141,2,210,173,243,6,
141,244,6,208,9,141,252,6,141,0,210,14
1,2,210,76,98,228
60 RESTORE 62:FOR J=1536 TO 1749:READ
A:POKE J,A:NEXT J
62 DATA 104,160,0,162,0,169,7,76,92,22
8
64 DATA 81,20,68,40,60,20,53,30,50,10,
53,20,60,40,72,20,91,30,81,10,73,20,68
,40,81,20,81,30,85,10,81,20,72,40
65 DATA 85,20,108,40,81,20,68,40,60,20
,53,30,50,10,53,20,60,40,72,20,91,30,8
1,10,72,20,68,30,72,10,81,20,85,30
66 DATA 96,10,91,20,81,40,81,20,0,0,53
,40,60,20,53,20,81,20,81,20,91,20,81,2
0,68,20,72,20,85,20,81,60,81,20
67 DATA 53,80,53,20,81,20,68,20,53,20,
60,20,68,20,60,80,60,20,53,20,50,40,50
,20,53,20,60,40,68,20,60,20,53,20
68 DATA 53,20,60,20,68,20,81,60,81,20,
53,80,53,20,81,20,68,20,53,20,60,20,72
,20,68,80,0,0,100,5,80,5,60,5,40,5
69 DATA 20,5,0,0,20,5,40,5,60,5,80,5,1
00,5,0,0,100,2,0,2,140,2,0,2,100,2,0,2
,140,2,0,0
80 A=ADR(VBI$):B=INT(A/256):C=A-256*B:
POKE 1538,C:POKE 1540,B:POKE 53761,174
:POKE 53763,170:POKE 1779,5
82 A=USR(1536):POKE 1788,88
90 GOSUB 1000
100 DIM NXTRM$(69),BRD$(48),BLD$(48),L
D$(48),RD$(48),P$(48),CC$(20),CO$(20),
T$(43),PD$(8),RN$(23),H$(23),U$(23)
109 REM * H$-Room"assoc. with chest co
ntents, CB$-Color of background *
110 DIM OBJ$(23),Q$(322),R$(345),CRY$(
14),CB$(23),M$(15),F$(37):RN$(23)=" ":
RESTORE 112:FOR J=1 TO 23:READ A
111 H$(J,J)=CHR$(A):NEXT J:FOR J=1 TO
23:READ A:CB$(J)=CHR$(A):NEXT J
112 DATA 3,22,12,15,18,16,20,19,1,2,4,
6,9,10,13,14,8,17,11,5,21,7,0
113 DATA 18,66,84,100,34,110,0,180,98,
178,4,16,174,178,52,20,6,0,254,210,64,
242,50
114 REM * Randomize room"names *
115 FOR J=1 TO 23
116 R=INT(RND(0)*23):FOR K=1 TO J:IF R
=ASC(RN$(K)) THEN 116
118 NEXT K:RN$(J,J)=CHR$(R):NEXT J:K=0
119 K=K+1:IF ASC(RN$(K))<>17 THEN 119
120 RN$(K,K)=RN$(18,18):RN$(18,18)=CHR
$(17)
121 REM * Put objects in proper place
*
122 FOR J=1 TO 23:K=ASC(H$(J)):OBJ$(J,
J)=RN$(K+1,K+1):NEXT J:CRY=ASC(OBJ$(22
)):OBJ$(22,22)=CHR$(17)
123 REM ** ALL STRING CHARACTERS IN LI
NES 124-170 ARE CTRL (EXCEPT THE X'S A
ND BLANKS) **
124 NXTRM$="APD,EBAFCGTBHE,FADGBECFKLI
DHMJINKGJOHSMNILOJMNVK5R,QQQPUTPLUVCRV
RSUOT"
125 BRD$="@P  OO@POOOOOOOOOOOOOOOOOOO
OOOOOOOOOOOONMOO  NM"
127 BLD$="  UVUVTTTTTTTTTTTTTTTTTTTTT
TTTTTTTTTTTTTTR5RS  "
130 LD$="  EFEFAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAABBCDCD  "
140 RD$="IJ  AAIJAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAGHBB  GH"
150 CC$="      WZZZLXXXXAXXXXK"
160 CO$="XBBBYWZZZLXXXXAXXXXK"
170 PD$="EFAAAAIJ"
179 REM * Room names *
180 Q$="###MDGH$XUQ####ZRRGHQ$VWDNH##5
HUIXPHG$RLO$JROGHQ$JREOHW##LURQ$FDOGUR
Q##ODFH$SLOORZV##UDZKLGH$ZKL5$"
181 Q$(99)="##FKLQD$GROO####5XU5OH$URE
H###PDJLF$ZDQG##LURQ##KRUVHVKRH##FKDL0#
PDLO##FR55HU$NHWWOH$"
182 Q$(183)="##V5\JODVV####VLOYHU$EULG
OH$FRORUHG$EDOOV$ZRRG$FURVVERZ$ERRN$RI
$NGRZQ###TXLOO$SHQ###EUDVV$EDODGFH$"
183 Q$(281)="$FDQGDHVWLFN###EDJ$RI$VDO
W####5UD\HU$UXJ##"
185 R$="$FDVWOH$FU\5W##YDP5LUH$V$ODLU$
5ULQFHVV$EDWK##GL0L0J$KDOO####ZLWFK$
V$GHQ###TXHHQ$V$EHGURRP"
186 R$(91)="$GDUN$GXQJHRQ##KHLU$V$QXUV
HU\$NLQJ$V$FKDPEHUVVRUFHUHU$V$GHQ$EODF
NVPLWK$VKR5"
187 R$(166)="$NQLJKW$V$URRP##FDVWOH$NL
WFKHQ##ZDWFKWRZHU####KRUVH$VWDEOH$##$MH
VWHU$V$URRP##FDVWOH$DUPRU\$"
188 R$(256)="################FDVWOH$LE
UDU\$DOFKHPLVW$V$ODE$FDVWOH$FKD5HO$###$
VWRUHURRP###PRQN$V$TXDUWHUV"
200 T$(1)=CHR$(0):T$(43)=CHR$(0):T$(2)
=T$:FLAG=0
209 REM * U$-Passage # blocked at cert
ain room *
210 RESTORE 212:FOR J=1 TO 23:READ A:U
$(J)=CHR$(A):NEXT J
212 DATA 0,0,0,23,0,0,0,0,12,0,0,18,31
,0,0,0,16,0,39,35,42,0,37
213 REM * Set up passages blocked *
214 FOR J=1 TO 23:A=ASC(U$(J)):IF A=0
THEN NEXT J
216 T$(A,A)=CHR$(1):NEXT J:T$(38,38)=C
HR$(1):T$(40,40)=CHR$(1):T$(43,43)=CHR
$(1)
300 ? :? "              PRE55 START";
302 IF PEEK(53279)<>6 THEN 302
304 ? CHR$(125):GOSUB 1100:GOSUB 1230
306 POSITION 6,0:? "carrying:":POSITIO
N 2,19:? "this is room # ":CUX=19:CUY=
11
310 RM=INT(RND(0)*8):PRRM=ASC(NXTRM$(R
M*3+INT(RND(0)*3+1))):A=RM+PRRM:IF ASC
(T$(A))>0 THEN 310
350 R1=ASC(NXTRM$(RM*3+1)):R2=ASC(NXTR
M$(RM*3+2)):R3=ASC(NXTRM$(RM*3+3))
356 IF R1=PRRM THEN RTRM=R2:LTRM=R3
357 IF R2=PRRM THEN RTRM=R3:LTRM=R1
358 IF R3=PRRM THEN RTRM=R1:LTRM=R2
359 POKE 1779,1:POKE 1788,198
360 POKE 559,0:POKE 712,ASC(CB$(ASC(RN
$(RM+1))+1)):POKE 710,48:POKE 77,0
361 POSITION 1,1:? "TO ROOM # ";LTRM;"
 ":POSITION 26,1:? "TO ROOM # ";RTRM;"
 "
362 POSITION 14,18:? "TO ROOM # ";PRRM
;" ":POSITION 17,19:? RM;" "
364 LT=ASC(T$(RM+LTRM)):RT=ASC(T$(RM+R
TRM)):GOSUB 1250
366 IF LT>0 THEN GOSUB 1215:GOTO 370
368 IF LT=0 THEN GOSUB 1210
370 IF RT>0 THEN GOSUB 1225:GOTO 382
372 IF RT=0 THEN GOSUB 1220
382 POKE 559,34:X=23:Y=19:Q=15*ASC(RN$
(RM+1))+1:M$=R$(Q,Q+14):GOSUB 1500:POS
ITION 22,19:? " ";
383 IF OBJ$(RM+1,RM+1)=RN$(RM+1,RM+1)
THEN POSITION 22,19:? CHR$(170)
384 X=24:Y=0:P=14*CRY+1:M$=Q$(P,P+13):
GOSUB 1500
390 POKE 712,ASC(CB$(ASC(RN$(RM+1))+1)
):POKE 710,48:POKE 77,0
392 POKE 209,18:POKE 208,14
400 S=STICK(0):IF STRIG(0)=0 THEN 410
402 IF S=14 THEN CUX=19:CUY=9
404 IF S=13 THEN CUX=19:CUY=16
406 IF S=7 THEN CUX=33:CUY=11
407 IF S=11 THEN CUX=6:CUY=11
410 LOCATE CUX,CUY,Z:POSITION CUX,CUY:
? "*";:TR=STRIG(0)
```

```
430 IF TR=0 AND CUY=16 THEN TEMP=RM:RM
=PRRM:PRRM=TEMP:GOTO 350
440 IF TR=0 AND LT=0 AND CUX=6 THEN LT
=RM+LTRM:PRRM=RM:RM=LTRM:GOTO 350
450 IF TR=0 AND RT=0 AND CUX=33 THEN R
T=RM+RTRM:PRRM=RM:RM=RTRM:GOTO 350
460 IF TR=0 AND CUY=9 THEN GOSUB 2000
470 IF FLAG=1 THEN FLAG=0:GOTO 360
480 IF FLAG=3 THEN POKE 1788,10:GOTO 6
00
490 POSITION CUX,CUY:? CHR$(Z);
500 GOTO 400
600 ? CHR$(125):POSITION 1,0:? "YOU FO
UND THE BOOK   congratulations!"
602 POSITION 2,19:? "start to repeat
 select for new game";
608 FOR J=0 TO 123:IF J>27 AND J<32 TH
EN NEXT J
610 F$(1)=CHR$(J):F$(37)=CHR$(J):F$(2)
=F$:FOR K=3 TO 16:POSITION 1,K:? F$;
612 A=PEEK(53279):IF A=6 THEN GRAPHICS
 0:POKE 756,START/256:GOTO 122
614 IF A=5 THEN GRAPHICS 0:POKE 756,ST
ART/256:? :? "Just a sec ...":GOTO 114
616 NEXT K:NEXT J:GOTO 608
999 REM * Change character set *
1000 DIM ZZ$(32):RESTORE 1010:FOR I=1
TO 32:READ A:ZZ$(I)=CHR$(A):NEXT I
1010 DATA 104,104,133,204,104,133,203,
104,133,206,104,133,205,162,4,160,0
1012 DATA 177,203,145,205,136,208,249,
230,204,230,206,202,208,240,96
1014 POKE 106,PEEK(106)-5:GRAPHICS 0:G
OSUB 1800:START=(PEEK(106)+1)*256:POKE
 756,START/256:POKE 752,1
1016 A=USR(ADR(ZZ$),57344,START):RESTO
RE 1020:FOR I=START+520 TO START+727:R
EAD A:POKE I,A:NEXT I
1018 RETURN
1020 DATA 170,170,170,170,170,170,170,
170,255,255,255,255,255,255,255,255,25
5,255,255,255,253,244,208,64
1030 DATA 253,244,208,64,0,0,0,0,0,0
,0,1,6,26,106,1,6,26,106,170,170,170,1
70
1040 DATA 127,31,7,1,0,0,0,0,255,255,2
55,255,127,31,7,1,64,144,164,169,170,1
70,170,170
1050 DATA 0,0,0,0,64,144,164,169,170,1
70,168,168,160,160,128,128,170,171,170
,174,170,186,170,234
1060 DATA 191,239,251,254,190,46,10,2,
191,47,11,2,0,0,0,0,191,239,251,254,19
0,238,250,254
1070 DATA 0,0,0,0,128,224,248,254,128,
224,248,254,190,238,250,254,254,251,23
9,191,190,184,160,128
1080 DATA 254,248,224,128,0,0,0,0,254,
251,239,191,190,187,175,191,0,0,0,0,2,
11,47,191
1090 DATA 2,11,47,191,190,187,175,191,
2,2,10,10,42,42,170,170,255,127,63,31,
15,7,3,1
1092 DATA 64,192,208,240,244,252,253,2
55,170,170,170,170,170,170,170,170
1099 REM * Change display list *
1100 DL=PEEK(560)+256*PEEK(561):POKE D
L+3,70:POKE DL+6,7:POKE DL+25,6:POKE D
L+26,7
1110 FOR J=0 TO 3:POKE DL+27+J,PEEK(DL
+29+J):NEXT J:FOR J=8 TO 23:POKE DL+J,
4:NEXT J:RETURN
1199 REM * Draw figures using strings
*
1200 FOR J=0 TO H:L=W*J:POSITION X,Y+J
:? P$(L+1,L+W):NEXT J:RETURN
1210 P$=LD$:H=11:W=4:X=4:Y=4:GOSUB 120
0:RETURN
1215 P$=BLD$:H=11:W=4:X=4:Y=4:GOSUB 12
00:RETURN
1220 P$=RD$:H=11:W=4:X=32:Y=4:GOSUB 12
00:RETURN
1225 P$=BRD$:H=11:W=4:X=32:Y=4:GOSUB 1
200:RETURN
1230 P$=CC$:H=3:W=5:X=17:Y=7:GOSUB 120
0:RETURN
1240 P$=CO$:H=3:W=5:X=17:Y=7:GOSUB 120
0:RETURN
1250 P$=PD$:H=0:W=8:X=16:Y=16:GOSUB 12
00:RETURN
1500 POSITION X,Y:FOR J=1 TO LEN(M$):?
 CHR$(ASC(M$(J))-3);:FOR K=1 TO 3:NEXT
 K:NEXT J:RETURN
1800 ? "Castle Hexagon gets its name f
rom":? "its six sided rooms with doors
 at"
1810 ? "alternate walls.  Rumor says a
n":? "ancient manuscript of wisdom"cal
led"
1820 ? "the 'Book of Known' lies hidde
n in":? "one of the castle's many room
s"
1830 ? "Your job is to find this treas
ure.":? "   Each room has a chest cont
aining"
1840 ? "an item associated with anothe
r":? "room.  You can only carry one it
em"
1850 ? "at a time.  Position the curso
r on":? "the chest and push 'fire'.  M
any"
1860 ? "doors have been bricked over b
ut":? "some will magically open when t
he"
1870 ? "chest contains the proper item
.":? "Position the cursor on the desir
ed"
1872 ? "door and push 'fire' to move b
etween":? "rooms.  The 'mat' in the mi
ddle"
1874 ? "represents the room you just c
ame":? "from.  Just remember you are f
acing"
1876 ? "into the room, (the map should
 help).":? "  A final warning ... it's
 not"
1878 ? "easy and if you re-run the pro
gram":? "all the rooms get switched ar
ound!";
1890 RETURN
1999 REM * Exchange chest contents *
2000 GOSUB 1240:TEMP=CRY:K=ASC(OBJ$(RM
+1)):P=14*K+1:CRY=K
2010 OBJ$(RM+1,RM+1)=CHR$(TEMP):X=24:Y
=0:M$=Q$(P,P+13):GOSUB 1500:POKE 1788,
174:FOR J=1 TO 100:NEXT J
2020 POKE 1788,186:GOSUB 1230
2025 IF CRY=17 THEN FLAG=3
2030 IF TEMP=ASC(RN$(RM+1)) THEN GOSUB
 2500
2040 RETURN
2499 REM * Clear passage *
2500 FOR J=1 TO 100:NEXT J:POKE 1788,1
0:FLAG=0:A=ASC(U$(RM+1)):IF A=0 THEN R
ETURN
2510 T$(A,A)=CHR$(0):FLAG=1:RETURN
```

MUSIC SOURCE LISTINGS

```
10 ; SOUND IN VBI ROUTINE
20 AUDC1 ="$D201 ;(53761)
30 AUDF1 ="$D200 ;(53760)
40 REST ="$6F3 ;(1779)
50 RESTCNT ="$6F4 ;(1780)
60 DURCNT ="$6F8 ;(1784)
70 STRPOS ="$6FC ;(1788)
80 AREA ="$600   ;(1536)
90  *=$0000
0100 TUNE1 LDX STRPOS ; Position is in
dex
0110  BEQ TUNE2 ; Not active
0120  LDA RESTCNT ; Pause active?
0130  BEQ TONE ; No
0140  DEC RESTCNT ; Yes
0150  BNE TUNE2 ; Pause not finished
0160  LDA AREA,X ; Frequency
0170  STA AUDF1 ; into channel 1
0180  ASL A ; 1 octave down
0190  NOP
0200  NOP
0210  STA AUDF1+2 ; into channel 2
0220  INX ; Next byte
0230  LDA AREA,X ; Duration
0240  BEQ FIN1   ; Quit if Dur ="0
0250  STA DURCNT
0260  INX ; index for next byte
0270  STX STRPOS ; Update string pos.
0280  BNE TUNE2  ; Skip over
0290 TONE DEC DURCNT ; Tone finished?
0300  BPL TUNE2 ; Not yet
0310  LDA #$00 ; Shut off tones
0320  STA AUDF1
0330  STA AUDF1+2
0340  LDA REST ; Initialize pause
0350  STA RESTCNT
0360  BNE TUNE2
0370 FIN1 STA STRPOS ; 0 in String pos
.
0380  STA AUDF1 ; and all sound off
0390  STA AUDF1+2
0400 TUNE2 JMP $E462
```

BALLOONS FIX

IN THE AUG/SEPT ISSUE, CHANGE THE LISTING OF THE BALLOONS PROGRAM: in Line 1600, FLAGG=0 not POKE FLAGG,0 as listed

HEXPOKE

The following program was part of Stan Ockers article last month on 'Machine Language Programming'.
    Sorry to have left it out

```
1 REM PROGRAM FOR "MACHINE LANGUAGE PR
OGRAMING, PART 1, NOV '82 ACE NEWSLETT
ER
10 REM ****************
20 REM **  HEXPOKE   **
30 REM **  S.O. 9-82  **
40 REM ****************
50 REM
100 DIM H$(16),D$(23),N$(5):H$="012345
6789ABCDEF":D$="@ABCDEFGHI!!!!!!!JKLMN
O"
110 OPEN #6,12,0,"S:":OPEN #1,4,0,"K:"
:POKE 752,1
300 ? CHR$(125);"INPUT ORIGIN IN HEX";
:INPUT N$:GOSUB 2010:AD=N:STRT=N:GOSUB
 1110
390 X=10:Y=4:GOSUB 910:AD=STRT:POSITIO
N 10,2:PRINT "STARTING ADDRESS ";STRT:
POSITION 10,22:? "DECIMAL VALUES"
400 GET #1,K
410 IF K=45 AND Y>4 THEN GOSUB 910:Y=Y
-2:GOSUB 910:AD=AD-8
420 IF K=61 AND Y<16 THEN GOSUB 910:Y=
Y+2:GOSUB 910:AD=AD+8
430 IF K=42 THEN GOSUB 910:GOSUB 510
440 IF K=43 THEN GOSUB 910:GOSUB 530
450 IF ((K)47) AND (K<58)) OR ((K)64)
AND (K<71)) THEN GOSUB 810
460 IF K=27 THEN GOTO 300
470 IF K=82 THEN A=USR(STRT):GOTO 300
500 POSITION 20,20:? PEEK(AD);"  ":POS
ITION 10,20:? AD:GOTO 400
```

SPELL by Ruth Ellsworth

Listing in PILOT

```
J"MEANS ESC CTRL CLEAR


10 R:SPELLING REVIEW
20 R:by Ruth Ellsworth
30 R:ACE NEWLETTER
40 R:3662 VINE MAPLE DRIVE
50 R:EUGENE, OREGON 97405
60 *START
70 T:TYPE IN 5 SPELLING WORDS
80 T:   ONE AT A TIME
90 A:$ONE
100 A:$TWO
110 A:$THREE
120 A:$FOUR
130 A:$FIVE
140 C:@B1373=16
150 C:@B1374=2
160 WRITE:S,     $ONE
170 WRITE:S,
180 WRITE:S,     $TWO
190 WRITE:S,
200 WRITE:S,     $THREE
210 WRITE:S,
220 WRITE:S,     $FOUR
230 WRITE:S,
240 WRITE:S,     $FIVE
250 PA:100
260 WRITE:S,)
270 C:#A=?\5
280 U(#A=0):*ONE
290 U(#A=1):*TWO
300 U(#A=2):*THREE
310 U(#A=3):*FOUR
320 U(#A=4):*FIVE
330 J:*SEC
340 *SEC
350 C:#B=?\5
360 J(#B=#A):*SEC
370 U(#B=0):*ONE
380 U(#B=1):*TWO
390 U(#B=2):*THREE
400 U(#B=3):*FOUR
410 U(#B=4):*FIVE
420 J:*THRI
430 *THRI
440 C:#C=?\5
450 J(#C=#A):*THRI
460 J(#C=#B):*THRI
470 U(#C=0):*ONE
480 U(#C=1):*TWO
490 U(#C=2):*THREE
500 U(#C=3):*FOUR
510 U(#C=4):*FIVE
520 J:*FOR
530 *FOR
540 C:#D=?\5
550 J(#D=#A):*FOR
560 J(#D=#B):*FOR
570 J(#D=#C):*FOR
580 U(#D=0):*ONE
590 U(#D=1):*TWO
600 U(#D=2):*THREE
610 U(#D=3):*FOUR
620 U(#D=4):*FIVE
630 J:*LAST
640 *LAST
650 C:#E=?\5
660 J(#E=#A):*LAST
670 J(#E=#B):*LAST
680 J(#E=#C):*LAST
690 J(#E=#D):*LAST
700 U(#E=0):*ONE
710 U(#E=1):*TWO
720 U(#E=2):*THREE
730 U(#E=3):*FOUR
740 U(#E=4):*FIVE
760 GR:QUIT
770 J:*START
780 E:
790 *ONE
800 POS:5,5
810 WRITE:S,$ONE
820 A:
830 WRITE:S,)
840 A:
850 M:$ONE
860 T:
870 TY:GOOD! $ONE IS SPELLED $ONE [GOO
D! should be in inverse lettering for
all words to make it stand out
880 TN:$ONE IS SPELLED $ONE
890 POS:5,5
900 WRITE:S,$ONE
910 PA:100
920 WRITE:S,)
930 E:
940 *TWO
950 POS:5,5
960 WRITE:S,$TWO
970 A:
980 WRITE:S,)
990 A:
1000 M:$TWO
1010 T:
1020 TY:GOOD! $TWO IS SPELLED $TWO
1030 TN:$TWO IS SPELLED $TWO
1040 POS:5,5
1050 WRITE:S,$TWO
1060 PA:100
1070 WRITE:S,)
1080 E:
1090 *THREE
1100 POS:5,5
1110 WRITE:S,$THREE
1120 A:
1130 WRITE:S,)
1140 A:
1150 M:$THREE
1160 T:
1170 TY:GOOD! $THREE IS SPELLED $THREE
1180 TN:$THREE IS SPELLED $THREE
1190 POS:5,5
1200 WRITE:S,$THREE
1210 PA:100
1220 WRITE:S,)
1230 E:
1240 *FOUR
1250 POS:5,5
1260 WRITE:S,$FOUR
1270 A:
1280 WRITE:S,)
1290 A:
1300 M:$FOUR
1310 T:
1320 TY:GOOD! $FOUR IS SPELLED $FOUR
1330 TN:$FOUR IS SPELLED $FOUR
1340 POS:5,5
1350 WRITE:S,$FOUR
1360 PA:100
1370 WRITE:S,)
1380 E:
1390 *FIVE
1400 POS:5,5
1410 WRITE:S,$FIVE
1420 A:
1430 WRITE:S,)
1440 A:
1450 M:$FIVE
1460 T:
1470 TY:GOOD! $FIVE IS SPELLED $FIVE
1480 TN:$FIVE IS SPELLED $FIVE
1490 POS:5,5
1500 WRITE:S,$FIVE
1510 PA:100
1520 WRITE:S,)
```

HEXPOKE (con't)

by Stan Ockers

```
510 X=X+3:AD=AD+1:IF X>31 THEN X=10:Y=
Y+2:IF Y>16 THEN Y=4:AD=STRT
520 GOSUB 910:RETURN
530 X=X-3:AD=AD-1:IF X<10 THEN X=31:Y=
Y-2:IF Y<4 THEN Y=4:X=10:AD=STRT
540 GOSUB 910:RETURN
800 REM ** SUBR. TO PUT NEW NUMBER IN
MEMORY **
810 N1=16*(ASC(D$(K-47))-64):POSITION
X,Y:? CHR$(K);:X=X+1:GOSUB 910
820 GET #1,K:IF K=126 THEN GOSUB 910:X
=X-1:GOSUB 910:GET #1,K:GOTO 810
824 IF ((K)47) AND (K(58)) OR ((K)64)
AND (K(71)) THEN 828
826 GET #1,K:GOTO 824
828 N2=ASC(D$(K-47))-64:N=N1+N2:POKE A
D,N
830 POSITION X,Y:? CHR$(K);:X=X+2:IF X
>32 THEN X=10:Y=Y+2:IF Y>16 THEN Y=4:A
D=STRT-1
890 AD=AD+1:GOSUB 910:RETURN
900 REM ** POS. AND ? INV. OF CHAR **
910 LOCATE X,Y,Z:POSITION X,Y:? CHR$(Z
+128);:RETURN
1000 REM ** PRINT DEC, # N IN HEX **
1010 M=4096:ZFLG=0:FOR I=1 TO 4:J=INT(
N/M):IF J>0 THEN ZFLG=1
1020 IF (J>0) OR (ZFLG=1) OR (I>2) THE
N ? H$(J+1,J+1);
1030 N=N-M*J:M=M/16:NEXT I:RETURN
1100 REM ** PRINT SCREEN OF MEM. DATA
**
1110 ? CHR$(125):FOR Y=4 TO 16 STEP 2:
POSITION 5,Y:N=AD:GOSUB 1010:POSITION
10,Y
1120 FOR K=0 TO 7:N=PEEK(AD):GOSUB 101
0:? " ";:AD=AD+1:NEXT K
1130 NEXT Y:RETURN
2000 REM ** CONVERT HEX IN N$ TO DEC.
# N **
2010 N=0:FOR I=1 TO LEN(N$):N=N*16+ASC
(D$(ASC(N$(I))-47))-64:NEXT I:RETURN
```

Addressing Mode

	absol.	zpage	immed.	absol,X
STA	EE	E6		9D
INC	CE	C6		
DEC	3 byte	2 byte		3 byte

Single Byte Instructions:

INX	E8	C8
DEX	CA	88

Branches:

BNE	D0
BEQ	F0
BPL	10
BMI	30

Branch on Not Equal to zero (Z=0)
Branch on EQual to zero (z=1)
Branch on PLus (neg. flag reset N=0)
Branch on MInus (neg. flag set N=1)

Additional OP-Codes
Table 1

MORE OCKERS MACHINE LANGUAGE

STRINGING ALONG WITH PILOT

by Ruth Ellsworth

Having felt tangled up in strings when I first began programming in BASIC, I was delighted to discover ATARI PILOT made strings easy to use and much easier to understand.

In ATARI PILOT a string is any group of characters used together as a group. They can be a series of letters, numbers, symbols, or words, and can be used in combination. Strings are stored in "string variables," chosen by the programmer and indicated by placing a "$" before the string name. For example $NAME could be a string variable in which a personal name will be stored. As an illustration with my children, we thought of strings as being like slots in a toy shelf. Each slot has a name, and any toy desired is placed in the assigned slot and retrieved by going to the slot named.

One does not need to dimension strings in PILOT. One needs only to use the A: command followed by the undefined string variable to allow a desired string to be assigned to that variable in a program. The simple spelling program at the end of this article illustrates this common use of string variables.

String values can be assigned in several ways. They can be assigned from the keyboard during the run of a program using the Accept (A) command or placed into the accept buffer without keyboard input by using the command A:= followed by the string variable.

The Compute (C) command assigns values to string variables in a program without keyboard input during programming so its value is set when the program is run. Thus in the program below, if one desires to place spelling words directly into strings $ONE, $TWO, etc. before the program is run, one changes the A:$ONE to C:$ONE = desired word.

Through the use of the Compute command strings can be combined to form new strings. This is done by giving the string variable name the value of other strings values in the program which may be combined in any way desired. We found this ability of the PILOT language especially useful in word and story games to from compound words or ideas.

It is also possible to use associated string values in Atari PILOT. My children decided associated values were like looking at a stack of dishes (yes, Virginia, all the boys in our house do dishes). We have a stack of plates which are stacked several sizes together to save space. The ones we used most often are slid under ones of a smaller size. They decided all the values for associated strings are placed together, one under the other. When the value is desired, its location is indicated by the use of repeated $ to show its location in the stack. Two dollar signs, for example, show the value was second from the top. The more $ the further down in the stack as indicated by the number of dollar signs. Thus, $FIRST = ONE places the value ONE on the shelf; $ONE = TWO places TWO under the associated value ONE; and $TWO = THREE places the associated value THREE under TWO. In this example $ONE = ONE, $$ONE = TWO, and $$$ONE = THREE. Those more accustomed to programming or familiar with FORTH may not like this particualr explanation, but it has worked well for us. I offer it only in the event someone else may find it useful. One may, instead, want to think of the $ sign as addresses and of going to suceeding addresses.

In addition to allowing string variables to be displayed in a program with the Type command, PILOT allows the variables to be dumped so their values may be displayed with the DUMP command, and to be changed using the VNEW$ command. I have included an example of the VNEW$ command in the spelling program which allows the program to run continuously without defaulting to regular text mode.

The program included here is simple, but one of the most used educational programs at our house. It allows the child to input 5 words at a time which are selected randomly through the use of the random number option in PILOT. The words are first displayed in the order entered, and then individually. The child selects the length of time the word is displayed before he attempts to spell it by pressing the return key so the screen can be cleared and he can try spelling it. The time could be controlled by placing a PA: command and a time number in place of the A: command in the program. Each word is displayed after the attempt whether it is spelled correctly or not to reinforce the correct spelling.

ATARI PILOT FOR BEGINNERS

A Review by Ruth Ellsworth

ATARI PILOT FOR BEGINNERS by Jim Conlan and Tracy Deliman with Dymax is the most comprehensive teaching tool for ATARI PILOT ! have seen. What it lacks in sense of humor and fun that **PICTURE THIS!** had it makes up in the completeness with which it covers the PILOT language. I discovered several things about PILOT which I had missed or overlooked, such as the fact the Turtle moves backward.

The book is divided into short chapters suitable for lessons which include a self-test with answers and a summary of the ideas presented in the chapter just completed. The illustrations are simple but well chosen, and it is well indexed allowing easy review and reference.

No assumptions are made as to the reader's knowledge of either computers or computer language. A brief description of the PILOT language is followed by an explanation of the use of the keyboard with special emphasis on the control keys. Sound, which is easier than graphics in ATARI PILOT, is presented first, then turtle graphics, followed by general programming skills.

Though every subject was well covered, I liked the approach to graphics found in David Thorburg's PICTURE THIS! better, in spite of being less comprehensive. If I were trying to learn about ATARI PILOT, I would study the first four chapters of ATARI PILOT FOR BEGINNERS then read PICTURE THIS! before completing the rest of ATARI PILOT FOR BEGINNERS. However, in the event only one book can be chosen, ATARI PILOT FOR BEGINNERS is definitely my choice.

Under The Tree

By Brian Dunn, author of Brian's Arcade

This month I am going to write an article on some of the games I like and think will be very good Christmas or Hanukkah presents.

Pac-Man and Centipede from Atari Inc.

Pac-Man and Centipede are both very good games and they both look almost exactly like the arcade versions. The graphics on Pac-Man are very close to the arcade version and the sound is identical. Centipede is also a very good duplication of the arcade version. The sound and graphics are extremely good and are very close to the arcade version of Centipede.

Both Pac-Man and Centipede are available from Atari Inc. or your local software dealer. They both come on a ROM cartridge and cost about $45.

Protector II from Synapse Software

Protector II is a very good space game. I like Protector II for a couple different reasons. Protector II has very good graphics — a lot better than Protector I. The game also has superb sound and great animation. The game comes on a disk for around $30. For a full review of Protector II read the last A.C.E. Newsletter under "Brian's Arcade".

Wizard Of Wor from Roklan Software

Wizard Of Wor is an outstanding arcade game which could also be called an arcade adventure. If you have ever seen the arcade game Wizard Of Wor and you like it you will love the home version because the arcade version and the home version are almost identical right down to the sound, graphics and mazes. Wizard of Wor is available on disk for $40 and on ROM cartridge for $45. For a full review of Wizard Of Wor read the next A.C.E. newsletter.

Rumors . . .

I've heard the author of Caverns of Mars has come out with a Caverns of MArs II. I heard it is really good but instead of scrolling down the screen you scroll to the right of the screen which means the game is kind of like Air Strike from English Software and is also like the arcade version of Scramble or Super Cobra.

I have also heard, from a reliable source, that Atari Inc. has the home computer rights to Donkey Kong and Donkey Kong Junior and both of those games are supposed to be best sellers next year, just wait. If Atari is going to release both of those games I think they should put both of those games on one cartridge if possible.

Up and Coming games which look good:

Miner 2049'er. Although I have not seen this game, from the advertisement in the latest Compute! magazine it looks like a very good game somewhat similar to Donkey Knog. It has 10 different screens in which you must get to the top in each of them. I am hoping to get a review copy of Miner 2049'er and if I do I will review it as soon as I can. Miner 2049'er is available from Big Five Software for $49.95. Miner 2049'er is available on ROM cartridge and is the first 16k ROM cartridge ever.

Well that's all for this month folks, so I want all of you to have a very HAPPY HOLIDAYS.

—Merry Christmas and Happy Hanukkah from Brian Dunn

Meet Ruth Ellsworth

The stereotype of computer hobbyists as esoteric whizkids is quickly fading as the personal home computer becomes ever more popular. Just look at Atari Computer Enthusiasts of Eugene: we have members from all walks of life and of all ages.

For example, meet Ruth Ellsworth, whose normal role is being the mother of five boys and one girl ranging is ages from 3 to 22. She and her husband, who teaches at Lane Community College, raise their family on a small farm outside Eugene, Oregon. Their lifestyle is quite different from many computer buffs. Following a do-it-yourself philosophy, they raise much of their own food and butcher their own meat. They built their house and sew their own clothes. Their 12-year-old son has his turn on the family Atari after he has milked the family cow.

Ruth became interested in micro-computers after seeing a public TV program on computers in education about 2 years ago. To Ruth, the computer seems a way to help her children learn at home: to reinforce what they learn at school and to fill in possible gaps. As an educator myself, I appreciate her concern from a different perspective. A student's attitude and success are very reflective of the interest shown by parents (as well as teachers) in what the child is doing at school. Ruth has the foresight to see the many teaching situations provided by the computer.

Education has been the occupation of Ruth's ancestors back into the 13th Century. She herself has participated in the formation of ACE's new Educational Researchers, an organization dedicated to the educational use of computers both in the school and at home.

ERACE is proud to announce a new compilation of programs Ruth has written in PILOT. **Programmed Inquiry, Learning Or Teaching** is a programming language designed with teacher and student in mind. It is an authoring language for educators who want to design their own lessons, as well as a learning language for students who want to learn to "teach" the computer to do things. Ruth has written several programs for her children which she now shares with the members of ACE. Included are several math drills in which the student has the choice of addition, subtraction, multiplication or division. My children especially like competing against themselves in the timed drills. My 3-year-old enjoys matching colors and coloring flowers and balls with simple movements of a joystick. An etching program in two versions allows the child to draw and fill-in colors. She's even included one dealing with music on a keyboard depicted graphically on the screen. Still in the works to be added is a spelling program.

If you have Atari's PILOT cartridge, these programs are for you. Ruth has laid some groundwork upon which many of you will build. I look for much more to be done in PILOT, both by Ruth Ellsworth and by you whose imaginations are sparked by her creativity.

—R. DeLoy Graham

BUMPAS REVIEWS

ADVENTURE INTERNATIONAL, Box 3435, Longwood, FL 32750 (no prices given) has two new pieces of software available. They are fully warranted for 1 year. If a defect occurs beyond 1 year they will replace the item for $5.

The first is **DISKEY** by Sparky Starks. The disk contains a number of utilities which will recover damaged disk information. You may examine and modify the information on the disk. You may be able to repair some files which you thought were lost.

The documentation comes in a digest-size booklet of 64 pages, and seems very complete. It includes a good discussion of Atari disk parameters before it gets to the use and "abuse" of DISKEY.

I've had a lot of fun with these utilities. I've never seen the "inside" of a disk before. So I've been using it mainly to explore around on some of the disks I have. I hope I learn enough about machine language to repair damaged disks by the time I need it.

The second AI disk is **BASIC ROUTINES** by Jerry White. This piece of software is crammed with dozens of short routines and utilities divided into 20 items on a menu-driven disk. Most of the items provide a basic foundation for many of the programming routines you will need to use in developing your own software.

There are also some utilities to use in editing and otherwise manipulating your BASIC programs. There is also a 64-page digest sized booklet with a chapter (and program listing) for each Menu item, as well as three additional chapters discussing methods to conserve memory, speed up your program, and using PEEKs and POKEs.

This work by Jerry White is presented with color & graphics, sound and text and may be valuable to anyone who wants to begin programming their own software.

BRIDGE MASTER

This program is available on disk for $21.95 from **DYNACOMP**, 1427 Monroe Ave., Rochester, NY 14618. It is an upgrade of their former Bridge 2.0. If you own the original, you may obtain the upgrade by sending them your original cassette or disk, together with $5 plus $2 handling. They were VERY prompt. I received the disk in what seemed less time than was required for my order to reach NY. I also sent off two pages of questions, and they were also very prompt with responses to the questions. Most of my problems with the program was due to my lack of understanding about Duplicate Bridge. I've never played Duplicate. The documentation tells you the program generates hands numbered from '0-999'. A numbered hand produces the same card distribution each time you select it. You may play the same hand as many times as desired. You may also switch positions in the same hand, and see how you may do on the other side. The documentation also tells you the '0' hand produces a random hand each time you select 0. I didn't find that to be true, and asked Dynacomp where the random hand could be found. Their response showed me the hands were actually numbered from 0.00 to 999.99. That's 100,000 hands, instead of 1000. Who needs another random hand generator? I hope I live long enough to play 100,000 hands (not likely!). The program keeps score and saves it to a disk file so you can come back to it any time. The menu lets you display the last hand played. So it you're playing the numbered hands sequentially, it's easy to pick up where you left off. There are no graphics — it's all text. With all the scoring and reading and writing to disk it seems slower than the 2.0 version it replaces. It has added some functions not covered by the 2.0. They've added pre-emptive openings, Blackwood and Stayman conventions. The program will also recognize demand bids and jump-shift responses. These are all nice improvements. Dynacomp says "sometimes it makes dumb plays!" I can second that! It makes them by your partner, as well as by your opponents, so maybe it evens out. Little things like trumping one's own high card, setting up an opponent's loser. The program makes you look like a very good Bridge player — you can win a lot. It's good practice for someone wanting to learn to play. I think I'm a pretty good player, and I'm having fun with it.

—Jim Bumpas

ERACE

In the last month we have made contact with three other user groups interested in educational software and applications.

Bill Nordstrom, from the Twin City Atari Interest Group (ESIG/TAIG), has sent us their first efforts in compiling a listing of available software and software manufacturers. They are cataloging this information using "Filemanager 800". Hopefully we will be able to interface with them to build a complete listing and avoid duplicating our efforts.

Bill says they are also soliciting information on educational software and will appreciate hearing from anyone.

We also heard from David Ahl at MACE. He will be sending some programs and information on programs they are using in the Michigan school system where they have decided to install Atari computers.

The user group from MIT has shown an interest in what we are doing and will be in touch with us.

The December 7 meeting begins the software evaluation criteria development. We discussed our concepts of program content and design.

Alice Erickson suggests we try to develop a software program based on our evaluation criteria which will allow us, or anyone, to more thoroughly and objectively evaluate and rate software. Evaluations will then be more consistent. The manufacturers and writers will then know how the programs are rated and what we as users expect to see in a program.

ERACE has taken on a formidable project and we can use any help you have to offer. If you have any information on educational software let us know.

DATA PERFECT/ FILEMANAGER +

REVIEW/COMPARISON

by Kirt E. Stockwell

The first major difference shows up when loading the program into the computer. Data Perfect loads quickly and in a straightforward manner. Filemanager+ loads quite slowly, with built-in pauses and hard-crashed sectors which give a disk drive fits. I personally dislike commercial software marketed this way. (Seems like abuse of the equipment.) There must be a better method of protecting copyright privileges.

Another item not escaping my notice is: Data Perfect resides entirely in memory, whereas Filemanager+ keeps only the kernel in RAM and must stop to load the various utility segments individually as they are to be used. The problem here is room. While Data perfect is written in machine language, FileManager+ appears to be written in Basic. This leads to another point I want to bring to your attention. Filemanager functions best on a two-drive system. This saves the user from having to swap data and program disks constantly. There is a provision made in the program to circumvent this problem. An option allows you to "initialize" your data disks. This feature writes parts of the program to your data disks, making the program more convenient to use and cutting down on the amount of free data space on the disk.

Lest it seem my intention here is to run FileManager+ into the ground, there are many other things which need saying. First of all, the documentation provided with FileManager+ is miles ahead of what passes for documentation with Data Perfect. FileManager+ documentation actually guides you step-by-step through examples of creating input and output formats as well as examples of how to use each utility incorporated into the program.

Some features are common to both programs. These include the ability to define field types such as: Alpha Numeric, Dollars & cents, repeating numbers, Computed numeric and computed Dollars & cents. FileManager also allows modification of a format even though there may be many records stored using an old format which is to be changed. Modification of a format is followed by the conversion of the database to fit the new format. This is an essentially automatic function. This does not, however, destroy either the old format or the old database. Functions similar to this can be used to combine databases created using two differing formats.

FileManager+ also states the program can take advantage of the AXLON RAMDISK.

Both programs store the record format on the disk which holds the actual data. Both also provide for backup copies of the format and the data. Data Perfect also provides backup of the program disk: The program is written on both sides of the disk as it is sent.

Data Perfect is inherently more powerful and flexible than FileManager+. This is not to say everybody who needs a database manager should run out and get Data Perfect. Here we have a classic example of a program so powerful and flexible most people can't use it. For those of you who are familiar with computer languages used on LARGE systems, such as IBM Mainframes, Data Perfect has many similarities to RPG (Report Program Generator). While not exactly a language, RPG requires learning many specific facts and certain routines in order to manipulate the program to accomplish what you need. Data Perfect is much like this. While you will need to study and fiddle and probably foul up some, the end product is a database tailored to your needs and as powerful as if you had paid a professional programmer to write it for you. The reporting capabilities are stupendous, again acting very much like RPG.

I urge SYNAPSE to hire someone to convert FileManager+ into Machine Language, and encourage LJK to hire a competent documentation specialist to write a full length book to explain Data Perfect.

Both programs are top notch, and both are capable of performing at a higher level than most users are apt to need. For those of you whose needs are relatively simple, I suggest the acquisition of FileManager+. For more complex reporting and data manipulation, Data Perfect may be more appropriate. Another item you may wish to take into consideration is the ability to merge Data Perfect and Letter Perfect files. At this time, one of the more potent considerations might be the amount of time you are willing to spend learning to use the database manager of your choice, and how complex your needs are likely to get.

Machine Language Programming No.2

Index registers, flags and branching

There were a couple of mistakes in the Machine Language Programming #1. The pound signs indicating immediate mode were omitted before $79 and $AA in Listing 2. Also, the words "Contents" and "Address" were reversed in Figure 1. The listing of D:HEXPOKE, omitted last time, appears in the centerfold of this issue.

Besides the accumulator the 6502 CPU also has two index registers called 'X' and 'Y'. They can easily be incremented (with single byte instructions INX or INY), or decremented (DEX,DEY). Normally they are used to modify the address involved in an operation. LDA $0600,X for example means load the accumulator with the byte at location $0600 MODIFIED BY X. If X is 0 the byte at $0600 will be used. If X is 3 the byte at $0603 is used while if X = $BA, the byte at $06BA will be used. To get numbers into the index registers there is a full compliment of load X and load Y instructions just as there are LDAs (immediate, absolute and zero page addressing modes).

To see the value of the index register suppose the address of the byte of the upper left hand corner of the screen is $7C40 and we want to put a line of 'A's' across the top of the screen. We could write a program to store an 'A' from the accumulator into each of the 40 bytes involved but it is much easier using X to modify the address and just change X:

```
LDA #$21                ; 'A' Use Table 9.6 change to hex
LOOP                    STA $7C40,X
INX                     ; Next position
                        Go back to loop 39 more times
```

A couple of problems present themselves however...
(1) we must be sure X starts at 0
(2) We must stop the looping process after 39 loops.

The first problem can be taken care of by a LDX #$00 before the looping starts, but what about the second? For that we must learn about FLAGS.

Flags indicate some action has taken place. There are a number of them in the CPU, (all in one register actually), and they are either set (value = 1) or reset (value = 0). Most operations set or reset the flags depending upon results of the operation. For example, the LDA instruction will set the ZERO FLAG if the accumulator gets loaded with a zero and reset it if it doesn't. It will set the NEGATIVE FLAG if a negative number gets loaded, and reset it when a positive number is loaded. We haven't talked about positive and negative numbers but numbers $00 thru $7F are positive while numbers $80 thru $FF are negative.

All load instuctions affect the zero and negative flags while store instructions don't affect any flags. The increment and decrement instructions also affect these same two flags and there are special instructions which are executed or not depending on the condition of the flags. These BRANCHING INSTRUCTIONS permit us to do conditional looping, looping only while a certain condition holds true. Let's re-write the 'A's' program using a conditional loop.

```
                10 ; Put A's across top of screen
                20 ;
0000            30          *=" $600
0600 68         40          PLA
0601 A227       50          LDX  #$27       ; 39 in decimal
0603 A921       60          LDA  #$21       ; an 'A'
0605 9D407C 70 LOOP         STA  $7C40,X    ; Put on screen
0608 CA         80          DEX             ; Next position
0609 D0FA       90          BNE  LOOP       ; Again if not
zero
060B D0FE       0100 HOLD   BNE  HOLD       ; So screen not
 cleared
```

A few things to notice: To be able to use our test of the zero flag we count down rather than up. The branch takes place each time DEX results in the zero flag being reset (Branch on Not Equal to zero, BNE). The 'A's will be printed from right to left rather than the normal way because the index is decrementing. The loop is done 39 times for hex values $27 thru $01, it is not done for X = 0, (hint: try BPL instead). It is not necessary to reload the accumulatior each time because the decrementing of X or branching does not affect the accumulator.

All branches are two byte instructions and they are relative branches which means the point they branch to is measured forward or backward relative to the branch location itself. Positive numbers ($00-$7F) branch forward to future code while negative numbers ($80-$FF) refer to branching backward to code already written. To see where the branch goes you have to count bytes in a particular manner. On a forward branch start counting (in hex) with byte one being the byte following the branch instruction (the next op-code). Count up to but not including the op-code to which you want to branch (remember the number after $0F is $10). For backward branches, start with the branch byte itself (the one you are trying to determine, not the op-code), begin with $FF and count backward up to and including the op-code of the instruction to which you want to branch. Remember $EF comes after $F0. The count is then used as the second byte of the branch operation. Listing 1 is a 'do-nothing' program which you can use to practice branches. In summary; branches depend on flags set or reset by the previous instruction(s).

Rather than use an index for counting it is sometimes convenient to use a memory location. There are increments and decrements for both the absolute and zero page addressing modes. Table 1 gives an update of op-codes covered in this session. Try re-writing the line of 'A's program using a zero page counter rather than index register X. There are only a few zero page locations free for our use. Appendix D of the Basic Manual shows these to be $CB to $D1.

Use 'Hexpoke' to find the address of the upper left corner of the screen, ($58 is low order and $59 is high order byte). Using this address in the STA ____ ____,X operation, load and run the 'A's program. Fast wasn't it? actually machine language instructions take only a few millionths of a second each so the whole program took far less than a thousandth of a second. You often have to keep this speed in mind because things can happen too fast!

Problem #3

First extend the 'A's program to fill 256 screen spaces with A's. Then try to fill the whole screen with A's. Use index Y to keep track of 3 increments of the high order byte of the screen address. You will have to create a program which modifies itself. This is not good practice but can be done until we show a better way to handle the problem. If you run into problems with a failure to fill when X is zero, try DEX, then STA then the branch. Remember STA affects no flags.

Problem #4

Try to use loops to get an idea of how fast machine language instructions execute. Create a loop within a loop, the outer loop changing once every time the inner one goes through 256 cycles, (65,536 inner cycles to go through the outer loop). Even this may not provide enough delay to time the process. If not, create a third outside loop of 20 or so repetitions. You can use X and Y registers, absolute or zero page locations for the increments or decrements.

—Stan Ockers

```
                0520 ;
                0530 ; Do Nothing Program"to Illustrate Br
anches
                0540 ;
0611            0550        *=" $0000
0000 A9CB       0560 START  LDA  #$CB
0002 D00A       0570        BNE  GETBYTE
0004 CA         0580 DNONE  DEX
0005 1003       0590        BPL  UPONE
0007 88         0600        DEY
0008 30FA       0610        BMI  DNONE
000A E8         0620 UPONE  INX
000B C8         0630        INY
000C F0F6       0640        BEQ  DNONE
000E BD0002     0650 GETBYTE LDA $0200,X
0011 85CB       0660        STA  $CB
0013 F0EB       0670        BEQ  START
0015 D0ED       0680        BNE  DNONE
```

```
                0110 ;
                0120 ;
                0130 ; ANSWERS TO PROBLEMS
                0140 ;
                0150 ; Problem #1: A really wierd clock
                0160 ; I find the upper left corner at $7C
40 (32K)
                0170 ;
060D            0180        *=" $600
0600 68         0190        PLA
0601 A512       0200 AGIN   LDA  $12
0603 8D407C     0210        STA  $7C40
```

```
0606 A513   0220        LDA  $13
0608 8D417C 0230        STA  $7C41
060B A514   0240        LDA  $14
060D 8D427C 0250        STA  $7C42
0610 4C0106 0260        JMP  AGIN
            0270 ;
            0280 ; Problem #2
            0290 ; For paddle controllers
            0300 ;
0613        0310        *=   $600
0600 68     0320        PLA
0601 A9AA   0330        LDA  #$AA      ; Pure tone, lo
udness 10
0603 8D01D2 0340        STA  $D201     ; IN AUDC1
0606 AD7002 0350 MORE   LDA  $270      ; Paddle 0
0609 8D00D2 0360        STA  $D200     ; in AUDF1
060C 4C0606 0370        JMP  MORE      ; forever
            0380 ;
            0390 ; For joystick controller
            0400 ;
060F        0410        *=   $600
0600 68     0420        PLA
0601 A9AA   0430        LDA  #$AA
0603 8D01D2 0440        STA  $D201
0606 AD7802 0450 REPT   LDA  $278      ; Joystick 0
0609 0A     0460        ASL  A
060A 0A     0470        ASL  A
060B 8D00D2 0480        STA  $D200
060E 4C0606 0490        JMP  REPT
            0500 ;
            0510 ;
```

The Game Writers' Column

by Jon Atack, Eugene A.C.E.

THE GAME WRITER'S COLUMN is a new column especially for Atari users who know the basics of machine language but don't know how to put everything together to get the smooth graphics and awesome sounds of the commercial games. Each month, we'll be going over techniques used in just about every arcade game on the market, including:

* Vertical Blank Interrupts
* Player-Missile movement routines
* Display Lists
* Display List Interrupts
* Sound and Music routines
* Special Effects in colors and sounds
* System Secrets used by game-makers

First of all, before you can write in assembly language, you should have an assembler/editor. All an assembler/editor does is take the assembly language you type in (called Source Code) and translate it into the machine language the Atari understands, also called Object Code. The one I use now, and the one I strongly recommend to anyone programming in machine language is the **SynAssembler**, written by Synapse Software. It requires a disk drive and 48K, but is the fastest and easiest to use assembler I have seen. After using the Atari Assembler/Editor cartridge and being frustrated by its bugs and snail-paced assembling speed, I set out looking for a faster assembler which is just as easy to use with perhaps a few more powerful features, and the SynAssembler is it. Whatever you do, DON'T buy the Atari Assembler/Editor cartridge. I did, and it was a mistake.

On to this month's topic: Vertical Blank Interrupts, referred to as VBI. VBI is the key to the smoothness of the graphics in commercial games. But first, a short explanation of vertical blank is in order. When the TV is drawing a picture on the screen, it starts at the top left corner and draws lines from left to right until it finishes at the bottom. Then it waits for a few microseconds before it starts to draw another picture. This period of time between drawing the screens is known as the Vertical Blank, when the TV isn't drawing anything on the screen. A Vertical Blank Interrupt is simply when the Atari interrupts what it's doing to run a program timed with the TV screen, so it is executed during the vertical blank of the TV. Vertical blank occurs every 1/60th of a second, or 60 times every second.

Why is this important? First of all, VBI is useful for routines which need to move at a regular pace, never speeding up or slowing down, such as music routines. Second, graphics changes can be done during the vertical blank to make sure they don't occur when the screen is drawing over them. This is how commercial games achieve their ultra-smooth graphics: By doing all animation and screen drawing during the vertical blank. The change comes on the screen neatly and smoothly without the image jerking or blurring. Another use is for monitoring user inputs, such as reading the joystick, so the responsiveness stays the same speed even if the main program slows down.

Implementing your own VBI routine is surprisingly easy. There are two RAM vectors for VBI: one for immediate VBI and one for deferred VBI. The only difference is immediate VBI runs before the Operating System's VBI routine, and deferred VBI runs after the O.S. VBI routine. Which to use? The immediate is better for graphics changes and general I/O, but if the VBI routine is very time-consuming in execution it has to be deferred, or else the screen jerks 60 times a second, giving unsightly effects. Also, deferred VBI is useful for writing to hardware registers after the O.S. VBI routine has written to them, giving you the last word, so to speak. For example, this is how to get rid of the annoying attract mode when your game is running.

Once you have decided whether your VBI should be immediate or deferred, you should place your routine in memory, link its termination to the regular VBI processing, and modify the appropriate RAM vector to point to it. Here's how:

1. Place your VBI routine into memory. At the end of it, have it do a JMP $E45F for immediate VBI or a JMP $E462 for deferred VBI.
2. Now, in your regular program, you must set up the RAM vectors so your VBI routine will be run.
3. Load the X-register with the Hi-byte of the address of the beginning of your VBI routine.
   Example: LDX #$40 if the address was $4080
4. Load the Y-register with the Lo-byte of the address of the beginning of your VBI routine.
   Example: LDY #$80 if the address was $4080
5. For immediate VBI, load the accumulator with a 6 (LDA #6). For a deferred, load the accumulator with a 7 (LDA #7).
6. Do a JSR $E45C. $E45C is an O.S. routine called SETVBV which changes the VBI RAM vectors $222,$223 and $224,$225 to point to your VBI routine. Control will be returned to your program within 1/60th of a second or less.

An example to set up an immediate VBI routine located at $600 (Page Six) follows...

```
100  LDX #6
110  LDY #0
120  LDA #6
130  JSR $E45C
```

...and your VBI routine will have to end with a JMP $E45F.

Your VBI should begin running in 1/60th of a second following these instructions. If you have any problems or questions, feel free to call or write. My number is (503) 344-8347 or 686-8357, and my address is:

—Jon Atack
1865 West 29th St.
Eugene, Oregon 97405

In closing, I have also included two sound routines from my new game Meteor Storm. One is in assembly language form and one is in BASIC, for people who don't have an assembler. Both use techniques which will be covered in a future column, and feel free to modify them or use them in your own games. Until next month, good luck and great gaming!

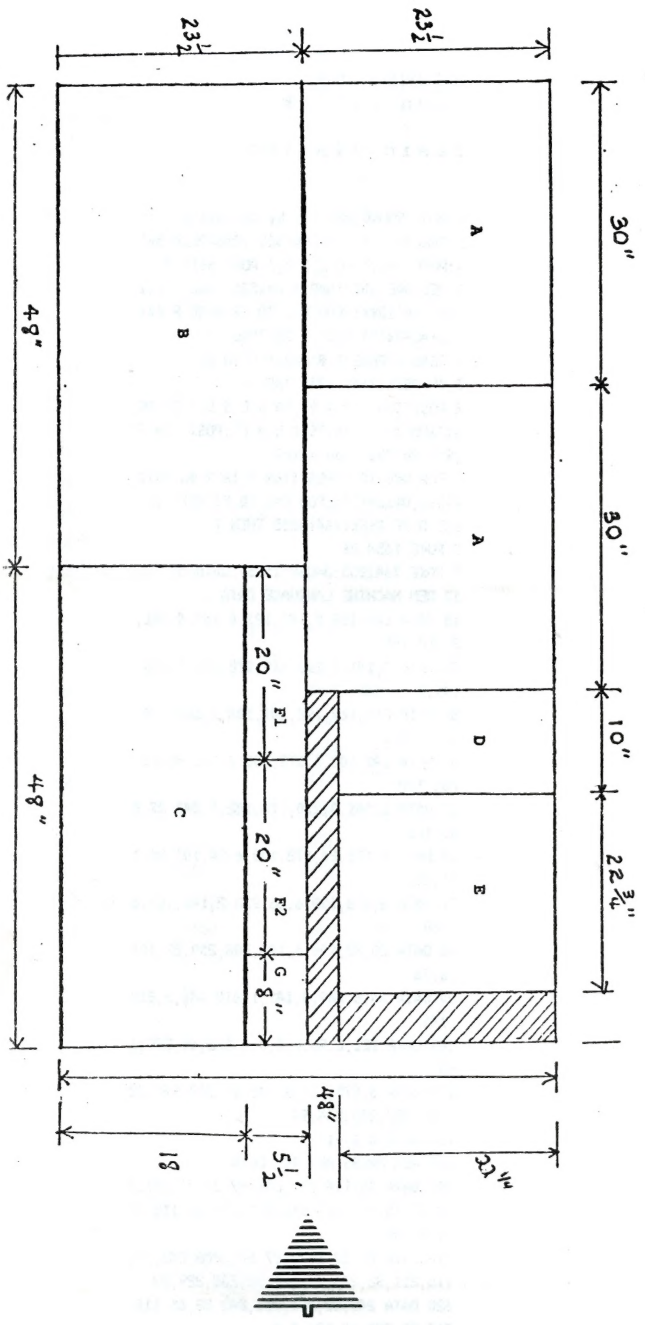
—Jon Atack

```
00001 ;  SOUNDS by Jon Attack
00100 ;    Special Sound Routine
00110 ;
00120 ;         by Jon Atack
00130 ;
00140 ; Written on the SynAssembler
00150 ;
00160 ; .HS 00  is equivalent to
00170 ; .BYTE 0 on the Atari Assembler
00180 ;
00190 ; Try changing the values of
00200 ; TIMER, INC, & RISE for
00210 ; different sound effects.
00220 ;
00230       .OR $600  STARTS AT $600
00240 START LDA #0
00250       STA SOUND
00260       LDA #$4
00270       STA $D208 INITIALIZE SOUND REGISTERS
00280       LDA #0
00290       STA $D207
00300       LDA #$AC
00310       STA $D205
00320       STA $D201
00330       LDA #192  TRY DIFFERENT NUMBERS HERE
00340       STA TIMER
00350       LDA #160  TRY DIFFERENT NUMBERS HERE
00360       STA INC
00370       STA RISE
00380 ;               SOUND ROUTINE
00390 LOOP  LDA INC
00400       STA $D204
00410       DEC INC
00420       STA $D200
00430       LDA INC
00440       BEQ END
00450       CMP RISE
00460       BCS GOON
00470       LDA RISE
00480       CLC
00490       ADC #$50
00500       STA INC
00510       LDA RISE
00520       SEC
00530       SBC #2
00540       STA RISE
00550       LDY #20
00560 LOOP3 JSR DELAY
00570       DEY
00580       BNE LOOP3
00590 GOON  JSR DELAY
00600       JMP LOOP
00610 END   LDA #0
00620       STA $D201
00630       STA $D205
00640       STA SOUND
00650       RTS
00660 INC .HS 00
00670 RISE .HS 00
00680 TIMER .HS 00
00690 COLOR .HS 00
00700 ;               DELAY AND CYCLE COLORS
00710 DELAY LDX #60
00720 LOOP4 INC COLOR
00730       LDA COLOR
00740       STA $D016
00750       DEX
00760       BNE LOOP4
00770       RTS
00780 SOUND .HS 00
```

SOUNDS by John Attack

Basic listing

```
1 REM  SOUND ROUTINE by Jon Atack
2 GRAPHICS 17:S=PEEK(560)+256*PEEK(561
):POKE S+8,7:POKE S+9,6:POKE S+10,7
3 RESTORE 600:TRAP 8:A=1535:POKE 752,1
:DIM A$(13*4):FOR D=1 TO 52:READ B:A$(
D,D)=CHR$(B):NEXT D:RESTORE
4 READ B:POKE A,B:A=A+1:GOTO 4
5 IF PEEK(764)<>255 THEN 8
6 POSITION 3,3:? #6;"A W E S O M E":PO
SITION 5,5:? #6;"S O U N D":POSITION 3
,9:? #6;"by  jon atack"
7 FOR D=0 TO 3:POSITION 3,13:? #6;A$(D
*13+1,D*13+13);:FOR E=1 TO 21:NEXT E:N
EXT D:IF PEEK(764)=255 THEN 7
8 POKE 1654,26
9 POKE 764,255:J=USR(1535):GOTO 5
10 REM MACHINE LANGUAGE DATA
15 DATA 104,169,0,141,121,6,169,4,141,
8,210,169
20 DATA 0,141,7,210,169,172,141,5,210,
141,1
30 DATA 210,169,192,141,104,6,169,160,
141,102,6
40 DATA 141,103,6,173,102,6,141,4,210,
206,102
50 DATA 6,141,0,210,173,102,6,240,37,2
05,103
60 DATA 6,176,26,173,103,6,24,105,80,1
41,102
70 DATA 6,173,103,6,56,233,2,141,103,6
,160
80 DATA 20,32,106,6,136,208,250,32,106
,6,76
90 DATA 36,6,169,0,141,1,210,141,5,210
,141
100 DATA 121,6,96,0,0,0,0,162,60,238,1
05
110 DATA 6,173,105,6,141,10,212,141,22
,208,202,208,244,96
120 DATA 0,0,-1
500 REM PRESS ANY KEY DATA
600 DATA 80,114,197,243,83,32,97,206,2
49,32,75,101,217,112,210,229,83,115,32
,193,238
610 DATA 89,32,107,197,249,208,242,69,
115,211,32,225,78,121,32,203,229,89
620 DATA 240,82,101,211,243,32,65,110,
217,32,235,69,121,0,0
```



TYPESET COPY
From Your Computer

If you have a modem, word processing and communications programs, you can transmit your manuscript, article, newsletter, or other text via telephone and have it converted into space-saving, easy-to-read type! Computer typesetting offers you high quality copy at "do-it-yourself" rates: just $2.00 per 1,000 characters. And it's easy to encode your copy with our designated typesetting commands. Hundreds of type styles to choose from with 8 styles and 12 sizes "on line." Write or call Dennis Hunt for a brochure and simple instructions.

EDITING & DESIGN SERVICES
30 East 13th Avenue Eugene, Oregon 97401
Phone (503) 683-2657

ATARI COMPUTER ENTHUSIASTS
3662 Vine Maple Dr. Eugene OR 97405

FIRST CLASS MAIL

Happy Holidays

Atari Computer Enthusiasts

A.C.E. is an independent computer club and user's group with no connection to the Atari Company, a division of Warner Communication Company. We are a group interested in educating our members in the use of the Atari Computer and in giving the latest News, Reviews and Rumors.

All our articles, reviews and programs come from you, our members.

Our membership is world-wide in scope; membership fees include the A.C.E. Newsletter. Dues are $10 a year for U.S., and $20 a year Overseas Airmail and include about 10 issues a year of the ACE Newsletter. **Subscription Dept: 3662 Vine Maple Dr., Eugene, OR 97405.**

President—Kirt Stockwell, 1810 Harris #139, Eugene, Or 97403 / 503-683-3005
Secretary—Charles Andrews, POB 1613, Eugene, Or 974401613 / 503-747-9892
Librarian—Chuck and Jody Ross, 2222 Ironwood, Eugene, OR 97401 503-343-5545
Editors—Mike Dunn, 3662 Vine Maple Dr., Eugene, Or 97405 / 503-344-6193
—Jim Bumpas, 4405 Dillard Rd., Eugene, Or 97405 / 503-484-9925
E.R.A.C.E. (Educational SIG) Editor—Larry Gold, 1927 McLean Blvd., Eugene, Or 97405 / 503-686-1490

Send a business-size SASE to the Ross' for the new, updated ACE Library List!!

Bulletin Board
(503) 343-4352

On line 24 hours a day, except for servicing and updating. Consists of a Tara equipped 48K Atari 400, 2 double-density disk drives, an Atari 825 printer, a Hayes SmartModem; running the ARMUDIC Bulletin Board software written by Frank C. Jones. See the Nov '82 issue for complete details.